Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Enhanced string covering ☆

Tomáš Flouri [a,1], Costas S. Iliopoulos [b,c,d], Tomasz Kociumaka [e], Solon P. Pissis [a,f,*,2], Simon J. Puglisi [b,3], W.F. Smyth [b,c,g,4], Wojciech Tyczyński [e]

[a] *Heidelberg Institute for Theoretical Studies, Germany*
[b] *King's College London, UK*
[c] *University of Western Australia, Australia*
[d] *Curtin University, Australia*
[e] *University of Warsaw, Poland*
[f] *Florida Museum of Natural History, University of Florida, USA*
[g] *McMaster University, Canada*

ARTICLE INFO

Article history:
Received 30 November 2012
Received in revised form 10 June 2013
Accepted 19 August 2013
Communicated by A. Apostolico

Keywords:
Periodicity
Quasiperiodicity
Covers
Seeds

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. If u is a cover of a superstring of y then u is a *seed* of y. Covers and seeds are two formalisations of *quasiperiodicity*, and there exist linear-time algorithms for computing all the covers and seeds 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: the *enhanced cover*; the *enhanced left cover*; and the *enhanced left seed*.

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. [1]). Periodicity is of paramount importance in many applications—for example, periodic factors in DNA are of interest to genomics researchers [2]—as well as in theoretical studies in combinatorics on words. Not long ago the term *regularity* [3] was coined to cover such variants, and a recent survey [4] provides coverage of the exact regularities so far identified and the sequential

☆ A preliminary version of this article appeared in the Proceedings of the Prague Stringology Conference 2012 (PSC 2012), pp. 75–88, 2012.

* Corresponding author at: Heidelberg Institute for Theoretical Studies (HITS), Schloss-Wolfsbrunnenweg 35, Heidelberg, D-69118, Germany.
E-mail address: solon.pissis@h-its.org (S.P. Pissis).

[1] Supported by the DFG grant STA 860/4.
[2] Supported by the NSF-funded iPlant Collaborative (NSF grant #DBI-0735191).
[3] Supported by a Newton Fellowship.
[4] Supported by a grant from the Natural Sciences & Engineering Research Council (NSERC) of Canada.

0304-3975/$ – see front matter
http://dx.doi.org/10.1016/j.tcs.2013.08.013

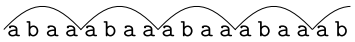

Fig. 1. Periodicity in string abaaabaaabaaabaaab.

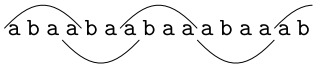

Fig. 2. Quasiperiodicity in string abaabaabaaabaaab.

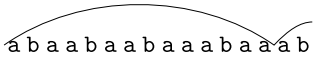

Fig. 3. Periodicity in string abaabaabaaabaaab.

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.

A string y is a *repetition* if $y = u^k$ for some non-empty string u of length m and some integer $k \geqslant 2$; 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 $x = \mathtt{abaababaaba}$, which is covered by copies of aba, yet not exactly periodic. To fill this gap, the idea of *quasiperiodicity* was introduced [5,6]. 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 $x = \mathtt{abaababaaba}$.

In [7], 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 [8] introduced the *minimal cover array* C—an array of size n of integers such that $\mathsf{C}[i]$, for all $0 \leqslant i < n$, gives the shortest cover of $y[0..i]$, or zero if no cover exists. Moreover, he described an on-line linear-time algorithm to compute C. In [9,10], 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 [11]. Finally, Li and Smyth [12] introduced the *maximal cover array* C^{M}—an array of size n of integers such that $\mathsf{C}^{\mathsf{M}}[i]$ gives the longest cover of $y[0..i]$, or zero if no cover exists—and showed that, analogous to the border array [13], 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 $x' = \mathtt{abaaababaaba}$ 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 $k \geqslant 2$ [15], even though an approximate solution could be computed in polynomial time [16].

Another well-known formalisation of quasiperiodicity is the *seed* of a string. A proper factor u of y is a *seed* of y if u is a cover of a superstring of y. Recently, Christou et al. introduced an intermediate notion between cover and seed: the *left seed* of a string [17]. A proper prefix u of y is a *left seed* of y if u is a cover of a superstring of y. Christou et al. also introduced the *minimal left-seed array* LS—an array of size n of integers such that $\mathsf{LS}[i]$ gives the shortest left seed of $y[0..i]$—and described a linear-time algorithm to compute it [17]. The *minimal right-seed array* was introduced and studied by Christou et al. in [18].

Our contribution. We have seen that while covers and seeds capture very well the repetitive nature of extremely repetitive strings, nevertheless most strings, and particularly those encountered in practice, will have no cover or seed, and so these measures of repetitiveness break down. More strings will have a useful k-cover, but this feature is hard to compute. Therefore we introduce new and more natural and applicable forms 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. 4).
- **Enhanced left cover.** A proper prefix u of a string y is an *enhanced left cover* of y, if u has at least two occurrences in y, and the number of letters of y which lie within some occurrence of u in y is a maximum over all such prefixes (see Fig. 5).
- **Enhanced left seed.** A proper prefix u of a string y is an *enhanced left seed* of y, if u has at least two occurrences in y, and the number of letters of y which lie within some occurrence of u in a superstring of y is a maximum over all such prefixes of y (see Fig. 6).

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013

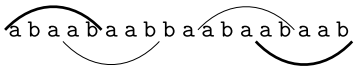

Fig. 4. Enhanced cover of string abaabaabbaabaabaab.

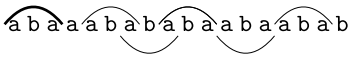

Fig. 5. Enhanced left cover of string abaaababababaabaabab.

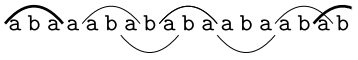

Fig. 6. Enhanced left seed of string abaaababababaabaabab.

These give rise to the following data structures.

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

In this article, we present efficient methods for computing the enhanced covers, the enhanced cover array, the enhanced left covers, the enhanced left-cover array, and the enhanced left seeds of a string. 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 enhanced covers of a string of length n in time $\mathcal{O}(n)$ and the enhanced cover array, the enhanced left covers, the enhanced left-cover array, and the enhanced left seeds 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 string 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 enhanced covers of y and an algorithm for computing the enhanced cover array of y. In Section 6, we present an algorithm for computing the enhanced left covers and the enhanced left-cover array of y. In Section 7, we present an algorithm for computing the enhanced left seeds of y. In Section 8, we present some experimental results. Finally, we briefly conclude with some future proposals in Section 9.

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 $0 \leqslant i < |x|$, the letter at index i of x. Each index i, for all $0 \leqslant i < |x|$, 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 $1 \leqslant k \leqslant n$. 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 $0 < p \leqslant |x|$, is called a *period* of x if $x[i] = x[i+p]$, for all $0 \leqslant i < |x| - p$. 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. [19]) that, for any non-empty string x $\mathsf{per}(x) + \mathsf{border}(x) = |x|$. 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 $0 \leqslant i < n$, 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 $u \neq y$ (see Fig. 7).

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013

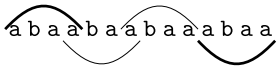

Fig. 7. Cover of string abaabaabaaabaa.

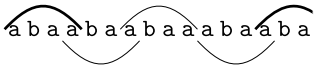

Fig. 8. Left seed of string abaabaabaaabaaba.

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 C[i], for all $0 \leqslant i < n$, 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 some occurrence of u in y is a maximum over all borders of y.

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 MEC[i], for all $0 \leqslant i < n$, stores the length of the minimal enhanced cover of the prefix $y[0..i]$ of y—zero if none.

Definition 4. A non-empty proper prefix u of a string y is an *enhanced left cover* of y if u has at least two occurrences in y, and the number of letters of y which lie within occurrences of u in y is a maximum over all such prefixes of y.

Definition 5. We define as *minimal enhanced left cover* the shortest enhanced left cover of y.

Definition 6. The *minimal enhanced left-cover array* MELC of size n of a non-empty string y of length n is the array of integers for which MELC[i], for all $0 \leqslant i < n$, stores the length of the minimal enhanced left cover of the prefix $y[0..i]$ of y—zero if none.

A string u is a *seed* of y, if it is a cover of a superstring of y. A left seed of a string y is a proper prefix of y that it is a cover of a superstring of y of the form yv, where v is a possibly empty string (see Fig. 8). The *minimal left seed* of y is the shortest left seed of y. The *minimal left-seed array* LS of size n of a non-empty string y of length n is the array of integers for which LS[i], $0 \leqslant i < n$, stores the length of the minimal left seed of the prefix $y[0..i]$ of y—zero if none.

Definition 7. A non-empty proper prefix u of a string y is an *enhanced left seed* of y if u has at least two occurrences in y and the number of letters of y which lie within occurrences of u in a superstring of y is a maximum over all such prefixes of y.

Definition 8. We define as *minimal enhanced left seed* the shortest enhanced left seed of y.

Definition 9. The *minimal enhanced left-seed array* MELS of size n of a non-empty string y of length n is the array of integers for which MELS[i], for all $0 \leqslant i < n$, stores the length of the minimal enhanced left seed of the prefix $y[0..i]$ of y—zero if none.

Example 10. Consider the string $y =$ abaaababaabaaaababaa. The following table illustrates the border array B of y, the minimal cover array C of y, the minimal enhanced cover array MEC of y, the minimal left-cover array MELC of y, the minimal left-seed array LS of y, and the minimal enhanced left-seed array MELS 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.

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013

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
B[i]	0	0	1	1	1	2	3	2	3	4	2	3	4	5	1	2	3	2	3	4
C[i]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MEC[i]	0	0	1	1	1	2	3	2	3	4	2	3	4	1	1	2	3	2	3	4
MELC[i]	0	0	1	1	1	1	3	2	3	3	2	3	3	3	1	1	3	3	3	3
LS[i]	0	0	2	3	4	4	4	6	6	6	9	9	9	9	14	14	14	16	16	16
MELS[i]	0	0	1	1	1	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3

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

Problem 1 *(Minimal enhanced cover). Compute the minimal enhanced cover of y.*

Problem 2 *(Minimal enhanced cover array). Compute the minimal enhanced cover array* MEC *of y.*

Problem 3 *(Minimal enhanced left cover). Compute the minimal enhanced left cover of y.*

Problem 4 *(Minimal enhanced left-cover array). Compute the minimal enhanced left-cover array* MELC *of y.*

Problem 5 *(Minimal enhanced left seed). Compute the minimal enhanced left seed of y.*

Problem 6 *(Minimal enhanced left-seed array). Compute the minimal enhanced left-seed array of y.*

3. Combinatorial properties of non-periodic borders

Definition 11. A string w is called *periodic* if it is non-empty and $2\mathsf{per}(w) \leqslant |w|$. 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, the minimal enhanced left cover, and the minimal enhanced left seed are never periodic.*

Proof. Let w be the minimal enhanced cover (resp. left cover, left seed) of a string y. Suppose w is periodic with longest border u. Then since $\mathsf{border}(w) + \mathsf{per}(w) \geqslant 2\mathsf{per}(w)$, it follows that $|u| \geqslant \mathsf{per}(w) \geqslant |w|/2$, and so u is a cover of w, hence also the minimal enhanced cover (resp. left cover, left seed) of x, a contradiction. □

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 a non-empty string 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 12. *The length of the kth shortest non-periodic border of a non-empty string y is at least $2^k - 1$. In particular, the total number of the non-periodic borders of y of length n is at most $\log n$.*

Proof. Let b_k be the length of the kth shortest non-periodic border of y. From Fact 2, $b_{k+1} \geqslant 2b_k + 1$. Moreover, $b_1 \geqslant 1$. Hence 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 $b_k \geqslant 2^k - 1 > n - 1$, which is clearly a contradiction. □

Note that by Corollary 12, 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 at 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 X[i], for all $0 \leqslant i < n$, stores the number of those non-periodic borders of the prefix $y[0..i]$ of y, which are simultaneously borders of y.

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013

Lemma 13. *The total number of occurrences of the non-periodic borders of y of length n is linear. More precisely*

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

Let us start with an auxiliary claim.

Claim 14. *Let $0 \leqslant i < j < n$ be integers. If $\mathsf{X}[i] > k$ then $i \geqslant 3 \cdot 2^k - 2$. Moreover, if $\mathsf{X}[i] > k$ and $\mathsf{X}[j] > k$ then $j - i \geqslant 2^k$.*

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 12, $|b| \geqslant 2^{k+1} - 1$. Any two occurrences of b in y must have their starting positions distant by at least $\frac{|b|+1}{2} \geqslant 2^k$. A pair of closer occurrences would induce a period of b no larger than $\frac{|b|}{2}$, which cannot 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 $i \geqslant |b| - 1 + 2^k \geqslant 3 \cdot 2^k - 2$. In the second part, there are occurrences of b ending at i and at j, which implies that $j - i \geqslant 2^k$. □

Proof of Lemma 13. 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

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

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

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

Therefore

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

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. [20,19,13]), and an array CB of size n such that, for all $0 \leqslant i < n$, $\mathsf{CB}[i]$ is 1 if $y[0..i]$ 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 15. Given a string y of length n, the *non-periodic border array* A is an array of size n of integers for which $\mathsf{A}[i]$, for all $0 \leqslant i < n$, stores the length of the longest non-periodic border of $y[0..i]$—zero if none.

Example 16. Consider the string $y = \texttt{aabaabaabbaaabaabaa}$. The following table illustrates the border array B, array CB, and the non-periodic border array A 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
CB[i]	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
A[i]	0	1	0	1	1	3	4	5	3	0	1	1	1	3	4	5	3	4	5

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013

The non-periodic border array A of a string y can be computed by Algorithm 1 in linear time. Algorithm 1 loops through the prefixes v of y and in each step considers two cases by Fact 2. If the longest border u of 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: NonperiodicBorderArray

Input : The border array B of string $y[0..n-1]$
Output: The non-periodic 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 17. Given a string y of length n, let R be an array of size n of integers for which R[i], for all $0 \leqslant i < n$, stores the length of the longest non-periodic border of $y[0..i]$, which is also a border of y—zero if none.

Example 18. Consider the string $y =$ aabaabaabbaaabaabaa. The following table illustrates the non-periodic border array A and array R 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
A[i]	0	1	0	1	1	3	4	5	3	0	1	1	1	3	4	5	3	4	5
R[i]	0	1	0	1	1	0	1	5	0	0	1	1	1	0	1	5	0	1	5

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 R[i] = A[i]. Otherwise, the border we seek is a shorter non-periodic border of v, so the result for v is the same as for u.

Algorithm 2: Array R

Input : The non-periodic 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 19. Given a string y of length n, PCP is an array of size n of integers for which PCP[i], for all $0 \leqslant i < n$, stores the number of letters of y which lie within an occurrence of the non-periodic prefix of length $i+1$ of y 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 non-periodic border array A of y. We also maintain an array LO of size n of integers, for which LO[i], for all $0 \leqslant i < n$, 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.

Example 20. Consider the string $y =$ aabaabaabbaaabaabaa. The following table illustrates array PCP 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
PCP[i]	13	0	15	14	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013

Algorithm 3: PositionsCoveredByPrefixesArray

Input : The non-periodic 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 non-periodic 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 21. *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 non-periodic border array A, and an inner while loop, iterating through the non-periodic borders of prefix $y[0..i]$. By Corollary 12, 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 22. Given a string y of length n, PCB is an array of size n of integers for which PCB$[i]$, for all $0 \leqslant i < n$, stores the number of letters of y which lie within an occurrence of the non-periodic prefix of length $i+1$ of y, which is a border of y—zero if the prefix is periodic or is not a border of y.

Example 23. Consider the string $y =$ `aabaabaabbaaabaabaa`. The following table illustrates array 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
PCB$[i]$	13	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5. Enhanced covers

In this section, we solve Problems 1 and 2; that is, computing the minimal enhanced cover and the minimal enhanced cover array. First, we show how to compute the minimal enhanced cover of string y. It can be computed by Algorithm 4. The algorithm takes as input the array R of y. It 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 13, the total number of occurrences of the considered borders is bounded by $2n$. Hence we obtain the following result.

Theorem 24. *The minimal enhanced cover of a string of length n can be computed in time* $\mathcal{O}(n)$.

Next, we show how to compute the minimal enhanced cover array MEC of string y. It can be computed by Algorithm 5. The algorithm takes as input the non-periodic border array A of y. Similarly as in the case of Algorithm 3, it goes through the non-periodic border array A, and iterates through the non-periodic 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 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 21, the PCP array can be computed in time $\mathcal{O}(n \log n)$. Hence we obtain the following result.

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013

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 non-periodic 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
```

Theorem 25. *The minimal enhanced cover array of a string of length n can be computed in time* $\mathcal{O}(n \log n)$.

6. Enhanced left covers

In this section, we solve Problems 3 and 4; that is, computing the minimal enhanced left cover and the minimal enhanced left-cover array. In particular, we only show how to compute the minimal enhanced left-cover array MELC of string y—trivially, MELC$[n-1]$ gives the minimal enhanced left cover of y. Array MELC can be computed by Algorithm 6. The algorithm takes as input the non-periodic border array A of y.

The variable *border* is used to ensure that there exists some prefix of y that has at least two occurrences in y (lines 4–5). In addition to the operations carried out by Algorithm 5 (lines 6–10), we check whether the number of positions covered by the currently considered border is greater than the number of positions covered by any previously considered border—of the current or of any previously considered prefix—and if that holds, we store the length of the current border in variable ℓ (lines 11–12). In case these numbers are equal, we store the length of the shortest border between the two (lines 13–14). Hence we obtain the following result.

Theorem 26. *The minimal enhanced left-cover array of a string of length n can be computed in time* $\mathcal{O}(n \log n)$.

7. Enhanced left seeds

In this section, we solve Problem 5; that is, computing the minimal enhanced left seed. We show how to compute the minimal enhanced left seed of string y. It can be computed by Algorithm 7. The algorithm starts with the computation

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013

Algorithm 6: MinimalEnhancedLeftCoverArray

Input : The non-periodic border array A[0..$n-1$] of string y[0..$n-1$]
Output: The minimal enhanced left-cover array MELC

```
1  PCP[0..n − 1] ← FillWithZeros; border ← 0
2  for i ← 0 to n − 1 do
3      b ← A[i]; ℓ ← 1
4      if border = 0 and b > 0 then
5          border ← 1
6      while b > 0 do
7          if PCP[b − 1] = 0 then
8              PCP[b − 1] ← min(2b, i + 1)
9          else
10             PCP[b − 1] ← PCP[b − 1] + min(b, i − LO[b − 1])
11         if PCP[b − 1] > PCP[ℓ − 1] then
12             ℓ ← b
13         else if PCP[b − 1] = PCP[ℓ − 1] and b < ℓ then
14             ℓ ← b
15         LO[b − 1] ← i
16         b ← A[b − 1]
17     if border = 1 then
18         MELC[i] ← ℓ
19     else
20         MELC[i] ← 0
```

of the PCP array and LO array, and the computation of array CB of y. Then the algorithm consists of a single for loop, iterating through the prefixes of y. By Definition 19, if the considered prefix is a border of y, the number of positions pc covered by it is given by PCP[i] (line 6). We also store the length of the longest (last) considered border by setting lb equal to $i+1$ (line 7). In case the considered prefix is not a border, the number of covered positions pc is equal to PCP[i] plus the minimum of two values: the length of the stored border and (in case these two factors overlap) the number of positions between the end of the last occurrence of the considered prefix and the end of the string. By Theorem 21, the PCP array can be computed in time $\mathcal{O}(n \log n)$. Trivially, the for loop requires linear time. Hence we obtain the following result.

Theorem 27. *The minimal enhanced left seed for a string of length n can be computed in time* $\mathcal{O}(n \log n)$.

Algorithm 7: MinimalEnhancedLeftSeed

Input : The non-periodic border array A[0..$n-1$] of string y[0..$n-1$]
Output: The length ℓ of the minimal enhanced left seed

```
1  Compute the PCP array and array LO
2  Compute array CB
3  lb ← 0; δ ← 0; ℓ ← 0
4  for i ← 0 to n − 1 do
5      if CB[i] > 0 then
6          pc ← PCP[i]
7          lb ← i + 1
8      else
9          pc ← PCP[i] + min(lb, n − 1 − LO[i])
10     if pc > δ then
11         δ ← pc
12         ℓ ← i + 1
```

Problem 6, that is, computing the minimal enhanced left-seed array, can be solved by applying the aforementioned algorithm for every prefix of y. However, since the computation of the PCP array is computed only once for the whole string, array MELS can be computed in time $\mathcal{O}(n^2)$.

8. Experimental results

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

Fig. 9 illustrates the maximal ratio of the total number of occurrences of the non-periodic borders of y, computed by Algorithm 4, 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 13. However, values close to this bound are not observed for small word length.

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013

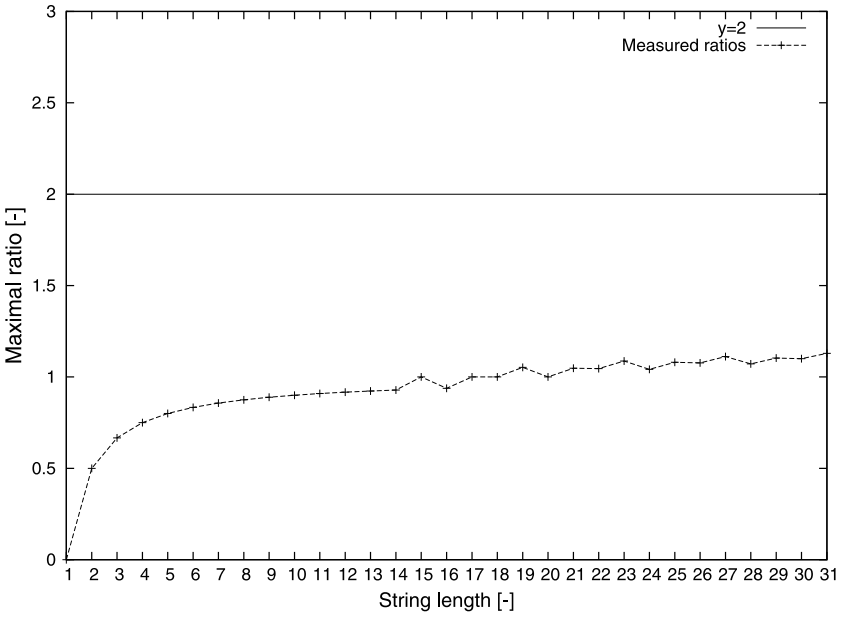

Fig. 9. Maximal ratio of the total number of occurrences of the non-periodic borders to the length n of string, for all strings on the binary alphabet.

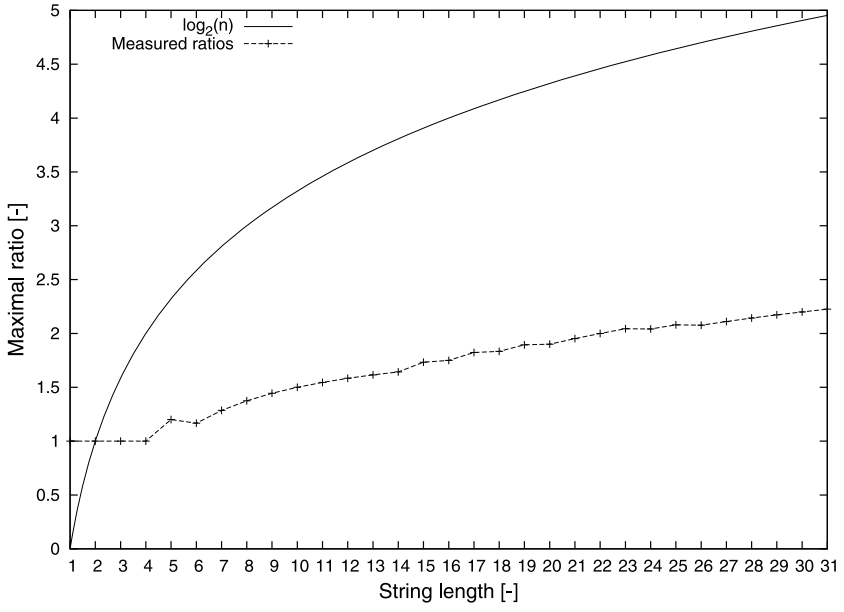

Fig. 10. Maximal ratio of the number of operations of Algorithm 5 to the length n of string, for all strings on the binary alphabet.

Figs. 10 and 11 illustrate the maximal ratio of the number of operations of Algorithm 5 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 21.

The main observation from Fig. 10 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. 11 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 5 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.

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013

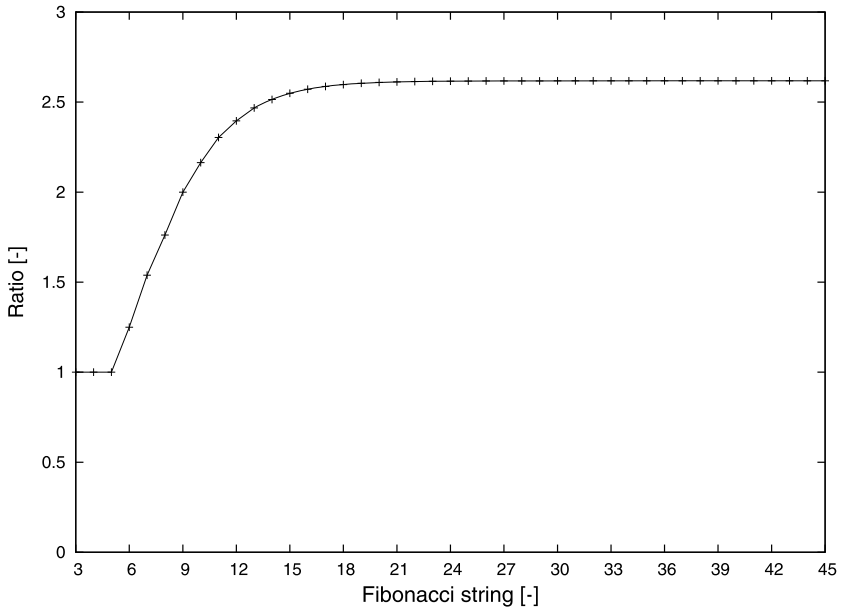

Fig. 11. Ratio of the number of operations of Algorithm 5 to the length n of string, for Fibonacci strings.

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

9. Concluding remarks

There are several directions for future work. Our immediate target is to investigate whether there exists an $\Omega(n^2)$-time algorithm for computing the minimal enhanced left-seed array.

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 $0 \leqslant i < n$. This also requires no additional effort.

References

[1] M. Lothaire (Ed.), Applied Combinatorics on Words, Cambridge University Press, 2005.
[2] R. Kolpakov, G. Bana, G. Kucherov, mreps: efficient and flexible detection of tandem repeats in DNA, Nucleic Acids Res. 31 (2003) 3672–3678.
[3] C.S. Iliopoulos, M. Mohamed, L. Mouchard, W.F. Smyth, K.G. Perdikuri, A.K. Tsakalidis, String regularities with don't cares, Nord. J. Comput. 10 (2003) 40–51.
[4] W.F. Smyth, Computing regularities in strings: a survey, Eur. J. Comb. 34 (2013) 3–14.
[5] A. Apostolico, A. Ehrenfeucht, Efficient detection of quasiperiodicities in strings, Technical Report CSD-1R-I048, Purdue University, 1990.
[6] A. Apostolico, A. Ehrenfeucht, Efficient detection of quasiperiodicities in strings, Theor. Comput. Sci. 119 (1993) 247–265.
[7] A. Apostolico, M. Farach, C.S. Iliopoulos, Optimal superprimitivity testing for strings, Inf. Process. Lett. 39 (1991) 17–20.
[8] D. Breslauer, An on-line string superprimitivity test, Inf. Process. Lett. 44 (1992) 345–347.
[9] D. Moore, W.F. Smyth, An optimal algorithm to compute all the covers of a string, Inf. Process. Lett. 50 (1994) 239–246.
[10] D. Moore, W.F. Smyth, A correction to "An optimal algorithm to compute all the covers of a string", Inf. Process. Lett. 54 (1995) 101–103.
[11] C.S. Iliopoulos, K. Park, A work-time optimal algorithm for computing all string covers, Theor. Comput. Sci. 164 (1996) 299–310.
[12] Y. Li, W.F. Smyth, Computing the cover array in linear time, Algorithmica 32 (2002) 95–106.
[13] W.F. Smyth, Computing Patterns in Strings, Addison–Wesley, 2003.
[14] C.S. Iliopoulos, 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, 2008, pp. 64–73.
[15] R. Cole, C.S. Iliopoulos, M. Mohamed, W.F. Smyth, L. Yang, The complexity of the minimum k-cover problem, J. Autom. Lang. Comb. 10 (2005) 641–653.
[16] C.S. Iliopoulos, M. Mohamed, W.F. Smyth, New complexity results for the k-covers problem, Inf. Sci. 181 (2011) 2571–2575.
[17] M. Christou, M. Crochemore, C.S. Iliopoulos, M. Kubica, S.P. Pissis, J. Radoszewski, W. Rytter, B. Szreder, T. Walen, Efficient seed computation revisited, in: R. Giancarlo, G. Manzini (Eds.), Proceedings of the Twenty-Second Annual Symposium on Combinatorial Pattern Matching (CPM 2011), in: Lect. Notes Comput. Sci., vol. 6661, Springer, Italy, 2011, pp. 350–363.

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013

[18] M. Christou, M. Crochemore, O. Guth, C.S. Iliopoulos, S.P. Pissis, On the right-seed array of a string, in: B. Fu, D.-Z. Du (Eds.), Proceedings of the Seventeenth Annual International Computing and Combinatorics Conference (COCOON 2011), in: Lect. Notes Comput. Sci., vol. 6842, Springer, USA, 2011, pp. 492–502.
[19] M. Crochemore, C. Hancart, T. Lecroq, Algorithms on Strings, Cambridge University Press, USA, 2007.
[20] D.E. Knuth, J.H. Morris Jr., V.R. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (1977) 323–350.

Please cite this article in press as: T. Flouri et al., Enhanced string covering, Theoretical Computer Science (2013), http://dx.doi.org/10.1016/j.tcs.2013.08.013