Enhanced Covers of Regular & Indeterminate Strings using Prefix Tables

Ali Alatabbi[1], A. S. Sohidull Islam[2], M. Sohel Rahman*[1,3], Jamie Simpson[4,5], and W. F. Smyth**[2,5]

[1] Department of Informatics, King's College London
ali.alatabbi@kcl.ac.uk
[2] Algorithms Research Group, Department of Computing & Software
McMaster University
sohansayed@gmail.com,smyth@mcmaster.ca
[3] Department of Computer Science & Engineering
Bangladesh University of Engineering & Technology
msrahman@cse.buet.ac.bd
[4] Department of Mathematics and Statistics, Curtin University of Technology
Jamie.Simpson@curtin.edu.au
[5] School of Engineering & Information Technology
Murdoch University

Abstract. A ***cover*** of a string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is a proper substring $\boldsymbol{u}$ of $\boldsymbol{x}$ such that $\boldsymbol{x}$ can be constructed from possibly overlapping instances of $\boldsymbol{u}$. A recent paper [12] relaxes this definition — an ***enhanced cover*** $\boldsymbol{u}$ of $\boldsymbol{x}$ is a border of $\boldsymbol{x}$ (that is, a proper prefix that is also a suffix) that covers a *maximum* number of positions in $\boldsymbol{x}$ (not necessarily all) — and proposes efficient algorithms for the computation of enhanced covers. These algorithms depend on the prior computation of the ***border array*** $\boldsymbol{\beta}[1..n]$, where $\boldsymbol{\beta}[i]$ is the length of the longest border of $\boldsymbol{x}[1..i]$, $1 \le i \le n$. In this paper, we first show how to compute enhanced covers using instead the ***prefix table***: an array $\boldsymbol{\pi}[1..n]$ such that $\boldsymbol{\pi}[i]$ is the length of the longest substring of $\boldsymbol{x}$ beginning at position i that matches a prefix of $\boldsymbol{x}$. Unlike the border array, the prefix table is robust: its properties hold also for ***indeterminate strings*** — that is, strings defined on *subsets* of the alphabet Σ rather than individual elements of Σ. Thus, our algorithms, in addition to being faster in practice and more space-efficient than those of [12], allow us to easily extend the computation of enhanced covers to indeterminate strings. Both for regular and indeterminate strings, our algorithms execute in expected linear time. Along the way we establish an important theoretical result: that the expected maximum length of any border of any prefix of a regular string $\boldsymbol{x}$ is approximately 1.64 for binary alphabets, less for larger ones.

* Supported in part by a Commonwealth Academic Fellowship and a ACU Titular Fellowship. Currently on a sabbatical leave from BUET.
** Supported in part by a grant from the Natural Sciences & Engineering Research Council (NSERC) of Canada.

1 Introduction

The concept of *periodicity* is fundamental to combinatorics on words and related algorithms: it is difficult to imagine a research contribution that does not somehow involve periods of strings. But periodicity alone may not be the best descriptor of a string; for example, $\boldsymbol{x} = abaababab$, a string of length $n = 9$, has period 7 and corresponding ***generator***[6] $abaabab$, but it could well be more interesting that every position but one in $\boldsymbol{x}$ lies within an occurrence of ab. In 1990 Apostolico & Ehrenfeucht [3] introduced the idea of quasiperiodicity: a ***quasiperiod*** or ***cover*** of a string $\boldsymbol{x}$ is a proper substring $\boldsymbol{u}$ of $\boldsymbol{x}$ such that any position in $\boldsymbol{x}$ is contained in an occurrence of $\boldsymbol{u}$; $\boldsymbol{u}$ is then said to ***cover*** $\boldsymbol{x}$, which is said to be ***quasiperiodic***. Thus, for example, $\boldsymbol{u} = aba$ is a cover of $\boldsymbol{x} = ababaaba$. Several linear-time algorithms were proposed for the computation of covers [4,8,17,18], culminating in an algorithm [16] to compute the ***cover array*** $\boldsymbol{\gamma}$, where $\boldsymbol{\gamma}[i]$ gives the length j of the longest cover of $\boldsymbol{x}[1..i]$. Since the longest cover of $\boldsymbol{x}[1..j]$ is also a cover of $\boldsymbol{x}[1..i]$, $\boldsymbol{\gamma}$ implicitly specifies all the covers of every prefix of $\boldsymbol{x}$. A recent paper [2] extends the computation of $\boldsymbol{\gamma}$ to "indeterminate strings" (see below for definition).

Even though the cover of a string can provide useful information, quasiperiodic strings are on the other hand infrequent among strings in general. Another approach to string covering was therefore proposed in [15]: a set $U_k = U_k(\boldsymbol{x})$ of strings, each of length k, is said to be a ***minimum k-cover*** of $\boldsymbol{x}$ if every position in $\boldsymbol{x}$ lies within some occurrence of an element of U_k, and no smaller set of k-strings has this property. Thus $U_2(abaababab) = U_2(ababaaba) = \{ab, ba\}$. In [10] the computation of U_k was shown to be NP-complete, though an approximate polynomial-time algorithm was presented in [14].

Recall that a ***border*** of $\boldsymbol{x}$ is a possibly empty proper prefix of $\boldsymbol{x}$ that is also a suffix: every nonempty string has a border of length zero. Recently the promising idea of an ***enhanced cover*** was introduced [12]; that is, a border $\boldsymbol{u}$ of $\boldsymbol{x} = \boldsymbol{x}[1..n]$ that covers a maximum number $m \leq n$ of positions in $\boldsymbol{x}$. Then the ***minimum enhanced cover*** $\texttt{mec}(\boldsymbol{x})$ is the *shortest* border $\boldsymbol{u}$ that covers m positions, and [12] presented an algorithm to compute $\texttt{mec}(\boldsymbol{x})$ in $\Theta(n)$ time. Thus for $\boldsymbol{x} = abaababab$, $\texttt{mec}(\boldsymbol{x}) = \texttt{ab}$. Further, on the analogy of the cover array defined above, the authors proposed the ***minimum enhanced cover array*** $\texttt{MEC}_{\boldsymbol{x}}$ — for every $i \in 1..n$, $\texttt{MEC}_{\boldsymbol{x}}[\texttt{i}] = |\texttt{mec}(\boldsymbol{x}[\texttt{1..i}])|$, the length of the minimum enhanced cover of $\boldsymbol{x}[1..i]$ — and showed how to compute it in $\mathcal{O}(n \log n)$ time. In this

[6] Notation and terminology generally follow [19].

paper we introduce in addition the CMEC array, where $\texttt{CMEC}[i]$ specifies the number of positions in $\boldsymbol{x}$ covered by the border of length $\texttt{MEC}[i]$. Thus, for example, $\texttt{MEC}_{\texttt{abaababab}} = \texttt{001123232}$ and $\texttt{CMEC}_{abaababab} = 002346688$.

In order to compute $\texttt{MEC}_{\boldsymbol{x}}$, the authors of [12] made use of a variant of the ***border array*** — that is, an integer array $\boldsymbol{\beta}[1..n]$ in which for every $i \in 1..n$, $\boldsymbol{\beta}[i]$ is the length of the longest border of $\boldsymbol{x}[1..i]$. In this paper we adopt a different approach to the computation of $\texttt{MEC}_{\boldsymbol{x}}$, using, instead of a border array, the ***prefix table*** $\boldsymbol{\pi} = \boldsymbol{\pi}[1..n]$, where for every $i \in 1..n$, $\boldsymbol{\pi}[i]$ is the length of the longest substring at position i of $\boldsymbol{x}$ that equals a prefix of $\boldsymbol{x}$. It has long been folklore that $\boldsymbol{\beta}$ and $\boldsymbol{\pi}$ are "equivalent", but it has only recently been made explicit [6] that each can be computed from the other in linear time. However, this equivalence holds only for ***regular*** strings $\boldsymbol{x}$ in which each entry $\boldsymbol{x}[i]$ is constrained to be a single element of the underlying alphabet Σ.

We say that a letter λ is ***indeterminate*** if it is any nonempty subset of Σ, and thus a string $\boldsymbol{x}$ is said to be ***indeterminate*** if some constituent letter $\boldsymbol{x}[i]$ is indeterminate. The idea of an indeterminate string was first introduced in [11] — with letters constrained to be either regular (single elements of Σ) or Σ itself — and the properties of these strings have been much studied by Blanchet-Sadri [7] and her collaborators as "partial words" or "strings with holes". Indeterminate strings can model DNA sequences on $\Sigma = \{A, C, G, T\}$ when ambiguities arise in determining individual nucleotides (letters).

Two indeterminate letters λ and μ are said to ***match*** (written $\lambda \approx \mu$) whenever $\lambda \cap \mu \neq \emptyset$, a relation that is in general nontransitive [13,22]: $a \approx \{a, b\}$ and $\{a, b\} \approx b$, but $a \not\approx b$. An important consequence of this nontransitivity is that the border array no longer correctly describes *all* of the borders of $\boldsymbol{x}$: it is no longer necessarily true, as for regular strings, that if $\boldsymbol{u}$ is the longest border of $\boldsymbol{v}$, in turn the longest border of $\boldsymbol{x}$, then $\boldsymbol{u}$ is a border of $\boldsymbol{x}$. On the other hand, the prefix array retains all its properties for indeterminate strings $\boldsymbol{x}$ and, in particular, correctly identifies all the borders of every prefix of $\boldsymbol{x}$ [6]. [20] describes algorithms to compute the prefix table of an indeterminate string; conversely, [9] proves that there exists an indeterminate string corresponding to every feasible prefix table, while [1] describes an algorithm to compute the lexicographically least indeterminate string determined by any given feasible prefix table. There is thus a many-many relationship between the set of all indeterminate strings and the set of all prefix tables. Consequently, computing $\texttt{MEC}_{\boldsymbol{x}}$ (or simply MEC when there is no ambiguity) from the prefix table $\boldsymbol{\pi} = \boldsymbol{\pi}_{\boldsymbol{x}}$

rather than from a variant of the border array allows us to extend the computation to indeterminate strings.

In Section 2 we outline the basic methodology and data structures used to compute the minimum enhanced cover array from the prefix table, while illustrating the ideas with an example. Then Section 3 provides a proof of the algorithm's correctness, as well as an analysis of its complexity, both worst and average case. In Section 4 we discuss the practical application of our algorithms, in terms of time and space requirements, and compare our prefix-based implementation with the border-based implementation of [12]. Section 5 extends the enhanced cover array algorithm to indeteterminate strings (for rooted covers) and outlines various other extensions, particularly to generalizations of MECs.

2 Methodology

In this section we describe the computation of $\texttt{MEC}_{\boldsymbol{x}}$, the enhanced cover array of $\boldsymbol{x}$, based on the prefix array $\boldsymbol{\pi}$. Since every minimum enhanced cover of $\boldsymbol{x}$ is also a border of $\boldsymbol{x}$, we are initially interested in the covers of prefixes of $\boldsymbol{x}$. For this purpose we need arrays whose size is $\texttt{B}$, the maximum length of any border of any prefix of $\boldsymbol{x}$. Noting that $\texttt{B}$ must be the maximum entry in the prefix array $\boldsymbol{\pi}$, we write $\texttt{B} = \max_{2 \le \texttt{i} \le \texttt{n}} \boldsymbol{\pi}[\texttt{i}]$.

Definition 1 *In the* ***maximum no cover*** *array* $\texttt{MNC} = \texttt{MNC}[1..\texttt{B}]$*, for every* $q \in 1..\texttt{B}$*,* $\texttt{MNC}[q] = q'$*, where* q' *is the maximum integer in* $1..q$ *such that the prefix* $\boldsymbol{x}[1..q']$ *has no cover — that is, such that* $\boldsymbol{\gamma}[q'] = 0$*.*

As shown in Figure 1, once $\texttt{B}$ is computed in $\Theta(n)$ time from the prefix array $\boldsymbol{\pi}$, $\texttt{MNC}$ can be easily computed in $\Theta(\texttt{B})$ time using the cover array $\boldsymbol{\gamma}[1..\texttt{B}]$ of $\boldsymbol{x}[1..\texttt{B}]$. Note that the entries in $\texttt{MNC}$ are monotone nondecreasing with $1 \le \texttt{MNC}[q] \le q$ for every $q \in 1..\texttt{B}$. The following is fundamental to the execution of our main algorithm:

Observation 2 *If a prefix* $\boldsymbol{v} = \boldsymbol{x}[1..q]$ *of* $\boldsymbol{x}$ *has a cover* $\boldsymbol{u}$*, then* $\boldsymbol{v} \neq \texttt{mec}(\boldsymbol{x})$ *(since* $|\boldsymbol{u}| < q$ *and* $\boldsymbol{u}$ *covers every position covered by* $\boldsymbol{v}$*).*

Thus $\texttt{MNC}[\texttt{q}]$ specifies an upper bound $q' \in 1..q$ on the length of a minimum enhanced cover of $\boldsymbol{x}$. Two other arrays are required for the computation, both of length $\texttt{B}$:

Definition 3 *For every* $q \in 1..\texttt{B}$*:*

- $\texttt{PR}[\texttt{q}]$ *is the rightmost position in* $\boldsymbol{x}$ *at which the prefix* $\boldsymbol{x}[1..q]$ *occurs;*

procedure Compute_MNC($n, \boldsymbol{\pi}$; B, $\boldsymbol{\gamma}$, MNC)
B $\leftarrow \boldsymbol{\pi}[2]$
for $i \leftarrow 3$ **to** n **do**
    B $\leftarrow \max(\mathtt{B}, \boldsymbol{\pi}[\mathtt{i}])$
▷ *Compute* $\boldsymbol{\gamma}[1..\mathtt{B}]$ *of* $\boldsymbol{x}[1..\mathtt{B}]$ *using*
▷ *the algorithm Compute_PCR of [2].*
Compute_PCR(B, $\boldsymbol{\pi}$; $\boldsymbol{\gamma}$)
▷ *Note that* MNC *can overwrite* $\boldsymbol{\gamma}$.
for $q \leftarrow 1$ **to** B **do**
    if $\boldsymbol{\gamma}[q] = 0$ **then** MNC$[q] \leftarrow q$
    else MNC$[q] \leftarrow$ MNC$[q-1]$

Fig. 1. Computing MNC from the prefix array $\boldsymbol{\pi}[1..n]$ and the cover array $\boldsymbol{\gamma}[1..B]$.

- CPR[q] *is the number of positions in* $\boldsymbol{x}$ *covered by occurrences of* $\boldsymbol{x}[1..q]$.

Here is an example of the arrays introduced thus far:

	1	2	3	4	5	6	7	8	9	10
$\boldsymbol{x}$ =	a	b	a	b	a	a	b	a	b	a
$\boldsymbol{\pi}$ =	10	0	3	0	1	5	0	3	0	1
$\boldsymbol{\gamma}$ =	0	0	0	2	3					
MNC =	1	2	3	3	3					
PR =	10	8	8	6	6					
CPR =	6	8	10	8	10					
MEC =	0	0	1	2	3	1	2	3	2	3
CMEC =	0	0	2	4	5	4	6	8	8	10

Note that for $\boldsymbol{x}[1..9]$ and $\boldsymbol{x}[1..10]$, there are actually *two* borders that cover a maximum number of positions; in each case the border of minimum length is identified in MEC.

The algorithm Compute_MEC is shown in Figure 2. In the first stage, B and MNC are computed and the arrays CMEC, PR and CPR are initialized. Then every position $i > 1$ such that $q = \boldsymbol{\gamma}[i] > 0$ is considered. Using MNC, the longest prefix $\mathtt{Q}' = \boldsymbol{x}[1..q']$ of $\boldsymbol{x}[1..q]$ that does *not* have a cover is identified; for prefixes of $\boldsymbol{x}[1..q]$ that do have a cover, the appropriate PR and CPR values have already been updated. There are two main steps in the processing of $\mathtt{Q}'$:

- Since i has now become the rightmost occurrence of $\mathtt{Q}'$ in $\boldsymbol{x}[1..i]$, we must set PR$[q'] \leftarrow i$ and increment the corresponding number CPR$[q']$ of positions covered.

```
procedure Compute_MEC(π; MEC, CMEC)
n ← |π|
Compute_MNC(n, π; B, γ, MNC)
MEC ← 0^n; CMEC ← 0^n; PR ← 1^B
for q ← 1 to B do CPR[q] ← q
for i ← 2 to n do
    q ← π[i]
▷ x[i..i+q−1] = x[1..q].
    while q > 0 do
▷ x[1..q′] is the longest prefix of x[1..q] without a cover.
        q′ ← MNC[q]
▷ x[1..q′] also occurs at i: update CPR[q′] & PR[q′].
        if i−PR[q′] < q′ then
            CPR[q′] ← CPR[q′]+i−PR[q′]
        else
            CPR[q′] ← CPR[q′]+q′
        PR[q′] ← i
▷ Update CMEC & MEC accordingly for interval i..i+q′−1.
        if CPR[q′] ≥ CMEC[i+q′−1] then
            MEC[i+q′−1] ← q′
            if CPR[q′] > CMEC[i+q′−1] then
                CMEC[i+q′−1] ← CPR[q′]
        q ← q′−1
```

Fig. 2. Computing MEC amd CMEC from the prefix array $\boldsymbol{\pi}$.

- If the number $\texttt{CPR}[q']$ of positions covered by occurrences of $\texttt{Q}'$ exceeds $\texttt{CMEC}[i+q-1]$, then CMEC and MEC must be updated accordingly.

These steps are repeated recursively for the longest proper prefix of $\texttt{Q}'$ that does not have a cover.

3 Correctness & Complexity of Compute_MEC

We begin by proving the correctness of Compute_MEC, which depends on the prior computation of $\boldsymbol{\pi} = \boldsymbol{\pi}_{\boldsymbol{x}}$ [6]. Consider first procedure Compute_MNC, where B is computed, followed by the cover array $\boldsymbol{\gamma}[1..\texttt{B}]$. Then for every $q \in 1..\texttt{B}$, $\texttt{MNC}[q] \leftarrow q$ whenever there is no cover of $\boldsymbol{x}[1..q]$, with $\texttt{MNC}[q] \leftarrow \texttt{MNC}[q-1]$ otherwise, an easy and straightforward calculation.

Compute_MEC then independently considers positions $i = 2, 3, \ldots, n$ for which $\boldsymbol{\pi}[i] > 0$; that is, such that a border of $\boldsymbol{x}$ of length $q = \boldsymbol{\pi}[i]$

begins at i. The internal **while** loop then processes in decreasing order of length the prefixes $\mathtt{Q'} = \boldsymbol{x}[1..q']$ of $\boldsymbol{x}[1..q]$ that have no cover — and that therefore, by Observation 2, can possibly be minimum enhanced covers of $\boldsymbol{x}[1..i{+}q'{-}1]$. Thus, for every $i \in 2..n$, all such borders $\boldsymbol{x}[1..q] = \boldsymbol{x}[i..i{+}q{-}1]$ are considered and, for each one, all such prefixes $\mathtt{Q'}$. For each q':

- the number $\mathtt{CPR}[\mathtt{q'}]$ of positions covered by $\mathtt{Q'}$ is updated, as well as the position $\mathtt{PR}[\mathtt{q'}] = \mathtt{i}$ of rightmost occurrence of $\mathtt{Q'}$;
- $\mathtt{MEC}[\mathtt{i{+}q'{-}1}]$ and $\mathtt{CMEC}[\mathtt{i{+}q'{-}1}]$ are updated accordingly for sufficiently large $\mathtt{CPR}[\mathtt{q'}]$.

We claim therefore that

Theorem 4 *For a given string $\boldsymbol{x}$, Compute_MEC correctly computes the minimum enhanced cover array $\mathtt{MEC}_{\boldsymbol{x}}$ and the number $\mathtt{CMEC}_{\boldsymbol{x}}$ of positions covered by it, based solely on the prefix array $\boldsymbol{\pi}_{\boldsymbol{x}}$.*

We have seen that in aggregate Compute_MEC processes a subset of the nonempty borders of every prefix $\boldsymbol{x}[1..i]$, devoting $\mathcal{O}(1)$ time to each one. As we have seen, each border $\mathtt{Q'}$ in each such subset is constrained to have no cover. We say that a string $\boldsymbol{v}$ is ***strongly periodic*** if it has a border $\boldsymbol{u}$ such that $|\boldsymbol{u}| \geq |\boldsymbol{v}|/2$; otherwise $\boldsymbol{v}$ is said to be ***weakly periodic***. Observe that the borders $\mathtt{Q'}$ must all be weakly periodic; if not, then they would have a cover $\boldsymbol{u}$ with $|\boldsymbol{u}| \geq |\boldsymbol{v}|/2$. In [12] the following result is proved:

Lemma 5 *There are at most $\log_2 n$ weakly periodic borders of a string of length n.*

It follows then that for each $i \in 2..n$, there are at most $\log_2 i$ borders considered, thus overall $\mathcal{O}(n \log n)$ time.

The space requirement of Compute_MEC, apart from the $\boldsymbol{\pi}$, MEC and CMEC arrays, each of length n, consists of three integer arrays (MNC (overwriting $\boldsymbol{\gamma}$), PR, CPR), each of length $\mathtt{B} < \mathtt{n}$. Thus

Theorem 6 *In the worst case, Compute_MEC computes* MEC *and* CMEC *from $\boldsymbol{\pi}$ using*

(*a*) $\mathcal{O}(n \log n)$ *time;*
(*b*) *three additional arrays* $1..\mathtt{B}$ *of integers* $1..n$*, thus* $\Theta(\mathtt{B} \log \mathtt{n})$ *bits of space.*

Now consider the expected (average) case behaviour of Compute_MEC. This depends critically on the expected length of the maximum border of

$\boldsymbol{x}[1..n]$; that is, the expected value of B. We show in the Appendix that for a given alphabet size, B approaches a limit as n goes to infinity. The limit is approximately 1.64 for binary alphabets, 0.69 for ternary alphabets, and monotone decreasing in alphabet size. Thus

Theorem 7 *In the average case, Compute_MEC requires $\mathcal{O}(n)$ time and $\Theta(\log n)$ additional bits of space.*

4 Comparing Border-Based and Prefix-Based Algorithms

As has been mentioned above, in order to compute $\texttt{MEC}_{\boldsymbol{x}}$, the authors of [12] made use of the ***border array***. On the other hand Compute_MEC is based on the ***prefix table***. We have already highlighted the advantage Compute_MEC has because of the use of a ***prefix table*** in lieu of a ***border array*** especially in the context of indeterminate strings. Additionally, the simplicity and low space usage of Compute_MEC encourage us to compare its practical performance with the algorithm of [12]. To this end this can be seen as a comparison between a border-based algorithm (i.e., the algorithm of [12]) for computing $\texttt{MEC}_{\boldsymbol{x}}$ and a prefix-based algorithm (i.e, Compute_MEC of the current paper) for doing the same. In what follows we will refer to the former algorithm as ECB and the latter as ECP.

We have implemented ECP (i.e., Compute_MEC) in C# using Visual Studio 2010. We got the implementation of ECB from the authors of [12]. However, ECB was implemented in C. To ensure a level playing ground, we re-implemented ECB in C# following their implementation. Then we have run both the algorithms on all binary strings of lengths 2 to 30. The experiments have been carried out on a Windows Server 2008 R2 64-bit Operating System, with Intel(R) Core(TM) i7 2600 processor @ 3.40GHz having an installed memory (RAM) of 8.00 GB. The results are illustrated in Figure 3 and 4, where the maximum number of operations carried out by each algorithm is reported in Figure 3. Figure 4 shows the ratio of the total number of operations performed by the Border-Based (ECB) [12] and Prefix-Based (ECP) algorithm to the length n of string, for all strings on the binary alphabet. As is evident from Figure 3 and 4 , ECP outperforms ECB and in fact it does show a linear behaviour verifying the claim in Theorem 7 above.

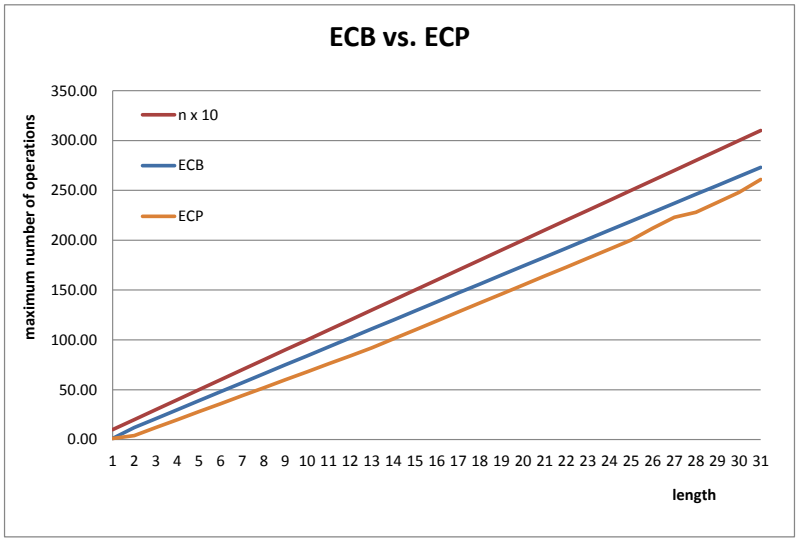

Fig. 3. The maximum number of operations performed by the Border-Based (ECB) [12] and Prefix-Based (ECP) algorithm (i.e., Compute_MEC) to compute the Minimum Enhanced Cover array, for all strings on the binary alphabet.

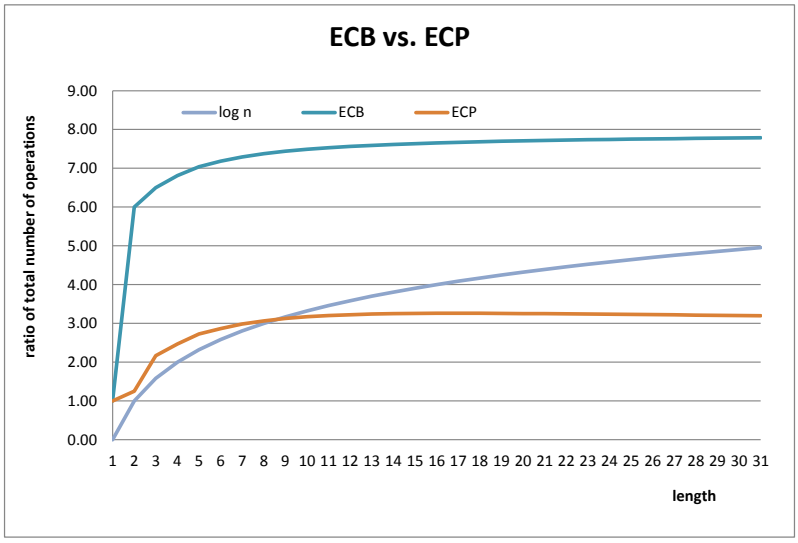

Fig. 4. Ratio of the total number of operations performed by the Border-Based (ECB) [12] and Prefix-Based (ECP) algorithm to the length n of string, for all strings on the binary alphabet.

5 Extensions

In Sections 2 and 3 we describe an algorithm to compute the minimum enhanced cover array $\texttt{MEC}_{\boldsymbol{x}}$ of a given string $\boldsymbol{x}$, based only on the prefix array $\boldsymbol{\pi}_{\boldsymbol{x}}$. As noted in the Introduction, since the prefix array can be computed also for indeterminate strings [20], this immediately raises the possibility of extending the MEC calculation to indeterminate strings.

In [2] two definitions of "cover" for an indeterminate string are proposed: a ***sliding cover*** where adjacent or overlapping covering substrings of $\boldsymbol{x}$ must match, and a ***rooted cover*** where each covering substring is constrained only to match a prefix of $\boldsymbol{x}$. The nontransitivity of matching (see Section 1) inhibits implementation of a sliding cover, but [2] shows how to compute all the rooted covers of indeterminate $\boldsymbol{x}$ from its prefix array in $\mathcal{O}(n^2)$ worst case time, $\Theta(n)$ in the average case. Thus it becomes possible to execute Compute_MNC for rooted covers, simply by replacing the function call to Compute_PCR by a function call to PCInd of [2]; that is, to compute the rooted cover array $\boldsymbol{\gamma}_{\boldsymbol{R}}[1..\texttt{B}]$, hence $\texttt{MNC}[1..\texttt{B}]$ and thus $\texttt{MEC}_{\boldsymbol{x}}$, all for indeterminate strings. Let us call this new algorithm Compute_MEC_Ind. We recall now a lemma from [5] stating that the expected number of borders in an indeterminate string is bounded above by a constant, approximately 29. Therefore, also for indeterminate strings, B can be treated as a constant, and we have the following remarkable result:

Theorem 8 *In the average case, Compute_MEC_Ind requires $\mathcal{O}(n)$ time and $\Theta(\log n)$ additional bits of space.*

We note further that the prefix array can be efficiently computed in a compressed form [20], taking advantage of the fact that for $i \in 1..n$, $\boldsymbol{\pi}[i] \neq 0$ if and only if $\boldsymbol{x}[i] = \boldsymbol{x}[1]$. Thus we can use two arrays POS and LEN to store nonzero positions in $\boldsymbol{\pi}$ and the values at those positions, respectively, thus saving much space in cases that arise in practice. We have designed a POS/LEN version of Compute_MEC that space restrictions do not allow us to describe here.

Finally, [12] describes extensions of the minimum enhanced cover array calculation, as follows:

- computation of the enhanced left-cover array of $\boldsymbol{x}$;
- computation of the enhanced left-seed array of $\boldsymbol{x}$.

Our prefix array approach yields efficient algorithms for these problems also, that may similarly be extended to rooted covers of indeterminate strings.

References

1. Ali Alatabbi, M. Sohel Rahman & W. F. Smyth, **Inferring an indeterminate string from a prefix graph**, *J. Discrete Algorithms* (2014) to appear.
2. Ali Alatabbi, M. Sohel Rahman & W. F. Smyth, **Computing covers using prefix tables**, `http://arxiv.org/abs/1412.3016`.
3. Alberto Apostolico & Andrzej Ehrenfeucht, *Efficient Detection of Quasi-periodicities in Strings*, Tech. Report No. 90.5, The Leonardo Fibonacci Institute, Trento, Italy (1990).
4. Alberto Apostolico, Martin Farach & C. S. Iliopoulos, Optimal superprimitivity testing for strings, *Inform. Process. Lett. 39-1* (1991) 17-20.
5. Md. Faizul Bari, Mohammad Sohel Rahman, Rifat Shahriyar, **Finding All Covers of an Indeterminate String in $O(n)$ Time on Average**. *Proc. Prague Stringology Conference* (2009) 263–271
6. Widmer Bland, Gregory Kucherov & W. F. Smyth, **Prefix table construction & conversion**, *Proc. 24th IWOCA*, Springer Lecture Notes in Computer Science LNCS 8288 (2013) 41–53.
7. Francine Blanchet-Sadri, *Algorithmic Combinatorics on Partial Words*, Chapman & Hall/CRC (2008) 385 pp.
8. D. Breslauer, An on-line string superprimitivity test, *Inform. Process. Lett. 44-6* (1992) 345-347.
9. Manolis Christodoulakis, P, J. Ryan, W. F. Smyth & Shu Wang,. **Indeterminate strings, prefix arrays & undirected graphs**, submitted for publication (2013).
10. Richard Cole, C. S. Iliopoulos, Manal Mohamed, W. F. Smyth & Lu Yang, **The complexity of the minimum k-cover problem**, *J. Automata, Languages & Combinatorics 10–5/6* (2005) 641–653.
11. Michael J. Fischer & Michael S. Paterson, **String-matching and other products**, *Complexity of Computation, Proc. SIAM-AMS 7* (1974) 113-125.
12. Tomáš Flouri, C. S. Iliopoulos, Tomasz Kociumaka, Solon P. Pissis, Simon J. Puglisi, W. F. Smyth & Wojciech Tyczyński, **Enhanced string covering**, *Theoret. Comput. Sci. 506* (2013) 102–114.
13. Jan Holub & W. F. Smyth, **Algorithms on indeterminate strings**, *Proc. 14th Australasian Workshop on Combinatorial Algs.* (2003) 36–45.
14. C. S. Iliopoulos, Manal Mohamed & W. F. Smyth, **New complexity results for the k-covers problem**, *Inform. Sciences 181* (2011) 2571–2575.
15. C. S. Iliopoulos & W. F. Smyth, **On-line algorithms for k-covering**, *Proc. Ninth Australasian Workshop on Combinatorial Algs.* (1998) 64–73.
16. Yin Li & W. F. Smyth, **An optimal on-line algorithm to compute all the covers of a string**, *Algorithmica 32–1* (2002) 95–106.
17. Dennis Moore & W. F. Smyth, An optimal algorithm to compute all the covers of a string, *Inform. Process. Lett. 50* (1994) 239-246.
18. Dennis Moore & W. F. Smyth, Correction to: An optimal algorithm to compute all the covers of a string, *Inform. Process. Lett. 54* (1995) 101-103.
19. Bill Smyth, *Computing Patterns in Strings*, Pearson Addison-Wesley (2003) 423 pp.
20. W. F. Smyth & Shu Wang, **New perspectives on the prefix array**, *Proc. 15th String Processing & Inform. Retrieval Symp.*, Springer Lecture Notes in Computer Science LNCS 5280 (2008) 133–143.
21. W. F. Smyth & Shu Wang, **A new approach to the periodicity lemma on strings with holes**, *Theoret. Comput. Sci. 410–43* (2009) 4295–4302.

22. W. F. Smyth & Shu Wang, **An adaptive hybrid pattern-matching algorithm on indeterminate strings**, *Internat. J. Foundations of Computer Science 20–6* (2009) 985–1004.

APPENDIX

We write $|\boldsymbol{x}|$ for the length of string $\boldsymbol{x}$. Here we show that the expected length of the longest border of a string $\boldsymbol{x}$ approaches a limit as $|\boldsymbol{x}|$ tends to infinity, the limit depending on the alphabet size. For a binary alphabet it is approximately 1.64. We use the following notation. $\sigma = |\Sigma|$ is the alphabet size, $B(w)$ is length of longest border of string w and $B_k(w)$ is length of longest border of string w which has length at most k (ie, ignoring any borders longer than k). Thus if $\boldsymbol{x} = babaababababbabaabab$ then $B(\boldsymbol{x}) = 8$ since $\boldsymbol{x}$ has longest border $babaabab$ and $B_4(\boldsymbol{x}) = 3$ since the longest border of $\boldsymbol{x}$ which has length at most 4 is aba. W_n is the set of all strings of length n on an alphabet of size σ. Since W_0 contains only the empty string we have $|W_0| = 1$.

Lemma 9 *The number of strings of length n on an alphabet of size σ which have a border of length k (not necessarily the longest border) is σ^{n-k}.*

Proof. A string with border of length k is periodic with period $n-k$ and so is determined by its length $n-k$ prefix. This prefix can be chosen in σ^{n-k} ways. □

We also need the following formula (which can be obtained using a computer algebra system).

Lemma 10 $\sum_{i=a}^{b} m\sigma^m = \frac{\sigma^{b+1}(\sigma\,(b+1)-\sigma-b-1)}{(\sigma-1)^2} - \frac{\sigma^a(a\sigma-a-\sigma)}{(\sigma-1)^2}$.

Clearly $|W_n| = \sigma^n$. The expected size of the longest border of a string of length n on an alphabet of size σ is therefore

$$\overline{B}(n) = \frac{1}{\sigma^n} \sum_{w \in W_n} B(w). \tag{1}$$

Similarly, the expected size of the longest border not exceeding k is

$$\overline{B}_k(n) = \frac{1}{\sigma^n} \sum_{w \in W_n} B_k(w). \tag{2}$$

Clearly $B(w) \geq B_k(w)$ so

$$\overline{B}(n) \geq \overline{B}_k(n). \tag{3}$$

Note that if $n \geq 2k$ then $W_n = \{uvw : u \in W_k, x \in W_{n-2k}, v \in W_k\}$ and so

$$\overline{B}_k(n) = \frac{1}{\sigma^n} \sum_{u \in W_k} \sum_{x \in W_{n-2k}} \sum_{v \in W_k} B_k(uxv). \tag{4}$$

Now $B_k(uxv) = B_k(uv)$ so if $n \geq 2k$,

$$\begin{aligned} \overline{B}_k(n) &= \frac{1}{\sigma^n} \sum_{u \in W_k} \sum_{v \in W_k} B_k(uv) \sum_{x \in W_{n-2k}} 1 \\ &= \frac{\sigma^{n-2k}}{\sigma^n} \sum_{u \in W_k} \sum_{v \in W_k} B_k(uv) \\ &= \frac{1}{\sigma^{2k}} \sum_{w \in W_{2k}} B_k(w) \\ &= \overline{B}_k(2k). \end{aligned} \tag{5}$$

With (3) we then have, for $n \geq 2k$,

$$\overline{B}(n) \geq \overline{B}_k(2k). \tag{6}$$

Now any border that is counted in the right hand side of (1) but not counted on the right hand side of (2) has length at least $k+1$. The sum of the lengths of such borders is, by Lemma 9,

$$\sum_{m=k+1}^{n} m\sigma^{n-m}.$$

So, by Lemma 10 and (5),

$$\begin{aligned} \overline{B}(n) &\leq \frac{1}{\sigma^n}\Big(\sum_{w \in W_n} B_k(w) + \sum_{m=k+1}^{n} m\sigma^{n-m}\Big) \\ &= \overline{B}_k(n) + \frac{1}{\sigma^n}\Big(\frac{\sigma^{n-k+1}k + \sigma^{n-k+1} - \sigma^{n-k}k - \sigma\, n - \sigma + n}{(\sigma-1)^2}\Big) \\ &< \overline{B}_k(n) + \frac{\sigma^{-k+1}k + \sigma^{-k+1} - \sigma^{-k}k - \sigma}{(\sigma-1)^2} \\ &= \overline{B}_k(2k) + O(k\sigma^{-k}). \end{aligned} \tag{7}$$

Thus for all $n \geq 2k$

$$\overline{B}_k(2k) \leq \overline{B}(n) \leq \overline{B}_k(2k) + O(\sigma^{-k})$$

so they're contained in an arbitrarily small interval. Call this interval I_k and define $J_1 = I_1$ and for $i \geq 2$ $J_i = I_i \cap J_{i-1}$. Then $J_1, J_2, \ldots$ is a sequence of nested intervals whose lengths have limit 0. By the Nested Intervals Theorem this means the limit of $\overline{B}_n$ exists.

Using (3) and (7) with $k = 11$ we find that $lim_{n\to\infty}\overline{B}(n)$ lies in the interval $(1.6356, 1.6420)$ for binary alphabets. For ternary alphabets using $k = 6$ the limit lies in $(0.6811, 0.6864)$.