Longest Unbordered Factor in Quasilinear Time

Tomasz Kociumaka
Institute of Informatics, University of Warsaw, Warsaw, Poland
kociumaka@mimuw.edu.pl
https://orcid.org/0000-0002-2477-1702

Ritu Kundu
Department of Informatics, King's College London, London, UK
ritu.kundu@kcl.ac.uk
https://orcid.org/0000-0003-1353-4004

Manal Mohamed
Department of Informatics, King's College London, London, UK
manal.mohamed@kcl.ac.uk
https://orcid.org/0000-0002-1435-5051

Solon P. Pissis
Department of Informatics, King's College London, London, UK
solon.pissis@kcl.ac.uk
https://orcid.org/0000-0002-1445-1932

Abstract

A *border* u of a word w is a proper factor of w occurring both as a prefix and as a suffix. The *maximal unbordered factor* of w is the longest factor of w which does not have a border. Here an $\mathcal{O}(n \log n)$-time with high probability (or $\mathcal{O}(n \log n \log^2 \log n)$-time deterministic) algorithm to compute the *Longest Unbordered Factor Array* of w for general alphabets is presented, where n is the length of w. This array specifies the length of the maximal unbordered factor starting at each position of w. This is a major improvement on the running time of the currently best worst-case algorithm working in $\mathcal{O}(n^{1.5})$ time for integer alphabets [Gawrychowski et al., 2015].

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases longest unbordered factor, factorisation, period, border, strings

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.70

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.09924`.

1 Introduction

There are two central properties characterising repetitions in a word –*period* and *border*– which play direct or indirect roles in several diverse applications ranging over pattern matching, text compression, assembly of genomic sequences and so on (see [3, 6]). A period of a non-empty word w of length n is an integer p such that $1 \leq p \leq n$, if $w[i] = w[i+p]$, for all $1 \leq i \leq n-p$. For instance, 3, 6, 7, and 8 are periods of the word `aabaabaa`. On the other hand, a border u of w is a (possibly empty) proper factor of w occurring both as a prefix and as a suffix of w. For example, ε, `a`, `aa`, and `aabaa` are the borders of $w =$ `aabaabaa`.

In fact, the notions of border and period are dual: the length of each border of w is equal to the length of w minus the length of some period of w. For example, `aa` is a border of the word `aabaabaa`; it corresponds to period $6 = |$`aabaabaa`$| - |$`aa`$|$. Consequently, the basic data structure of periodicity on words is the *border array* which stores the length of the longest border for each prefix of w. The computation of the border array of w was the fundamental concept behind the first linear-time pattern matching algorithm – given a word w (pattern), find all its occurrences in a longer word y (text). The border array of w is better known as the "failure function" introduced by Knuth, Morris, and Pratt [12]. It is well-known that the border array of w can be computed in $\mathcal{O}(n)$ time, where n is the length of w, by a variant of the Knuth-Morris-Pratt algorithm [12].

Another notable aspect of the inter-dependency of these dual notions is the relationship between the length of the maximal unbordered factor of w and the periodicity of w. A maximal unbordered factor is the longest factor of w which does not have a non-empty border; its length is usually represented by $\mu(w)$, e.g. the maximal unbordered factor is `aabab` and $\mu(w) = 5$ for the word $w =$ `baabab`. This dependency has been a subject of interest in the literature for a long time, starting from the 1979 paper of Ehrenfeucht and Silberger [9] in which they raised the question – at what length of w, $\mu(w)$ is maximal (i.e., equal to the minimal period of the word as it is well-known that it cannot be longer than that). This line of questioning, after being explored for more than three decades, culminated in 2012 with the work by Holub and Nowotka [11] where an asymptotically optimal upper bound ($\mu(w) \leq \frac{3}{7}n$) was presented; the historic overview of the related research can be found in [11].

Somewhat surprisingly, the symmetric computational problem – given a word w, compute the longest factor of w that does not have a border – had not been studied until very recently. In 2015, Kucherov et al. [15] considered this arguably natural problem and presented the first sub-quadratic-time solution. A naïve way to solve this problem is to compute the border array starting at each position of w and locating the rightmost zero, which results in an algorithm with $\mathcal{O}(n^2)$ worst-case running time. On the other hand, the computation of the maximal unbordered factor can be done in linear time for the cases when $\mu(w)$ or its minimal period is small (i.e., at most half the length of w) using the linear-time computation of unbordered conjugates [8]. However, as has been illustrated in [15] and [2], most of the words do not fall in this category owing to the fact that they have large $\mu(w)$ and consequently large minimal period. In [15], an adaptation of the basic algorithm has been provided with average-case running time $\mathcal{O}(n^2/\sigma^4)$, where σ is the alphabet's size; it has also been shown to work better, both in practice and asymptotically, than another straightforward approach that employs data structures from [14, 13] to query all relevant factors.

The currently fastest worst-case algorithm to compute the maximal unbordered factor of a given word takes $\mathcal{O}(n^{1.5})$ time; it was presented by Gawrychowski et al. [10] and it works for integer alphabets (alphabets of polynomial size in n). This algorithm works by categorising bordered factors into *short* borders and *long* borders depending on a threshold, and exploiting the fact that, for each position, the short borders are bounded by the threshold and the long borders are small in number. The resulting algorithm runs in $\mathcal{O}(n \log n)$ time on average. More recently, an $\mathcal{O}(n)$-time average-case algorithm was presented using a refined bound on the expected length of the maximal unbordered factor [2].

Our Contribution. In this paper, we show how to efficiently answer the Longest Unbordered Factor question using combinatorial insight. Specifically, we present an algorithm that computes the *Longest Unbordered Factor Array* in $\mathcal{O}(n \log n)$ time with high probability. The algorithm can also be implemented deterministically in $\mathcal{O}(n \log n \log^2 \log n)$ time. This array specifies the length of the maximal unbordered factor at each position in w. We thus improve on the running time of the currently fastest algorithm, which reports only the maximal unbordered factor of w and works only for integer alphabets, taking $\mathcal{O}(n^{1.5})$ time.

Structure of the Paper. In Section 2, we present the preliminaries, some useful properties of unbordered words, the algorithmic toolbox, and a formal definition of the problem. We lay down the combinatorial foundation of the algorithm in Section 3 and expound the algorithm in Section 4; its analysis is explicated in Section 5. We conclude this paper with a final remark in Section 6.

2 Background

Definitions and Notation. We consider a finite *alphabet* Σ of *letters*. Let Σ^* be the set of all finite words over Σ. The *empty* word is denoted by ε. The *length* of a word w is denoted by $|w|$. For a word $w = w[1]w[2]\mathinner{.\,.} w[n]$, $w[i\mathinner{.\,.} j]$ denotes the *factor* $w[i]w[i+1]\mathinner{.\,.} w[j]$, where $1 \le i \le j \le n$. The *concatenation* of two words u and v is the word composed of the letters of u followed by the letters of v. It is denoted by uv or also by $u \cdot v$ to show the decomposition of the resulting word. Suppose $w = uv$, then u is a *prefix* and v is a *suffix* of w; if $u \neq w$ then u is a *proper prefix* of w; similarly, if $v \neq w$ then v is a *proper suffix* of w. Throughout the paper we consider a non-empty word w of length n over a *general alphabet* Σ; in this case, we replace each letter by its rank such that the resulting word consists of integers in the range $\{1, \dots, n\}$. This can be done in $\mathcal{O}(n \log n)$ time after sorting the letters of Σ.

An integer $1 \le p \le n$ is a *period* of w if and only if $w[i] = w[i+p]$ for all $1 \le i \le n-p$. The smallest period of w is called the *minimum period* (or *the period*) of w, denoted by $\lambda(w)$. A word u ($u \neq w$) is a *border* of w, if $w = uv = v'u$ for some non-empty words v and v'; note that u is both a proper prefix and a suffix of w. It should be clear that if w has a border of length $|w| - p$ then it has a period p. Thus, the minimum period of w corresponds to the length of the *longest border* (or *the border*) of w. Observe that the empty word ε is a border of any word w. If u is the *shortest border* then u is the shortest *non-empty* border of w.

The word w is called *bordered* if it has a non-empty border, otherwise it is *unbordered*. Equivalently, the minimum period $p = |w|$ for an unbordered word w. Note that every bordered word w has a shortest border u such that $w = uvu$, where u is unbordered. By $\mu(w)$ we denote the maximum length among all the unbordered factors of w.

Useful Properties of Unbordered Words. Recall that a word u is a border of a word w if and only if u is both a proper prefix and a suffix of w. A border of a border of w is also a border of w. A word w is unbordered if and only if it has no non-empty border; equivalently ε is the only border of w. The following properties related to unbordered words form the basis of our algorithm and were presented and proved in [7].

▶ **Proposition 1** ([7]). *Let w be a bordered word and u be the shortest non-empty border of w. The following propositions hold:*

1. *u is an unbordered word;*
2. *u is the unique unbordered prefix and suffix of w;*
3. *w has the form $w = uvu$.*

▶ **Proposition 2** ([7]). *For any word w, there exists a unique sequence $(u_1, \cdots, u_k)$ of unbordered prefixes of w such that $w = u_k \cdots u_1$. Furthermore, the following properties hold:*

1. *u_1 is the shortest border of w;*
2. *u_k is the longest unbordered prefix of w;*
3. *for all i, $1 \le i \le k$, u_i is an unbordered prefix of u_k.*

The computation of the unique sequence described in Proposition 2 provides a unique *unbordered-decomposition* of a word. For instance, for w = `baababbabab` the unique unbordered-decomposition of w is `baa` · `ba` · `b` · `ba` · `ba` · `b`.

Longest Successor Factor (Length and Reference) Arrays. Here, we present the arrays that will act as a toolbox for our algorithm. The longest successor factor of w (denoted by lsf) starting at position i, is the longest factor of w that occurs at i and has at least one other occurrence in the suffix $w[i+1\,..\,n]$. The *longest successor factor array* gives for each position i in w, the length of the longest factor starting both at position i and at another position $j > i$. Formally, the longest successor factor array (LSF_ℓ) is defined as follows.

$$\mathsf{LSF}_\ell[i] = \begin{cases} 0 & \text{if } \ i = n, \\ \max\{k \mid w[i\,..\,i+k-1] = w[j\,..\,j+k-1\}, & \text{for } \ i < j \le n. \end{cases}$$

Additionally, we define the LSF-*Reference Array*, denoted by LSF_r. This array specifies, for each position i of w, the *reference* of the longest successor factor at i. The *reference* of i is defined as the position j of the last occurrence of $w[i\,..\,i+\mathsf{LSF}_\ell[i]-1]$ in w; we say i *refers to* j. Formally, LSF-*Reference Array* (LSF_r) is defined as follows.

$$\mathsf{LSF}_r[i] = \begin{cases} \mathit{nil} & \text{if } \ \mathsf{LSF}_\ell[i] = 0, \\ \max\{j \mid w[j\,..\,j+\mathsf{LSF}_\ell[i]-1] = w[i\,..\,i+\mathsf{LSF}_\ell[i]-1]\} & \text{for } \ i < j \le n. \end{cases}$$

Computation: Note that the longest successor factor array is a mirror image of the well-studied longest previous factor array which can be computed in $\mathcal{O}(n)$ time for integer alphabets [4, 5]. Moreover, in [4], an additional array that keeps a position of some previous occurrence of the longest previous factor was presented; such position may not be the leftmost. Arrays LSF_ℓ and LSF_r can be computed using simple modifications (pertaining to the symmetry between the longest previous and successor factors) of this algorithm[1] within $\mathcal{O}(n)$ time for integer alphabets.

▶ **Example 3.** Let w = `aabbabaabbaababbabab`. The associated arrays are as follows.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$w[i]$	a	a	b	b	a	b	a	a	b	b	a	a	b	a	b	b	a	b	a	b
$\mathsf{LSF}_\ell[i]$	5	6	5	4	3	4	3	4	3	2	1	4	3	2	1	3	2	1	0	0
$\mathsf{LSF}_r[i]$	7	14	15	16	17	10	11	14	15	18	19	17	18	19	20	18	19	20	nil	nil

▶ Remark. For brevity, we will use lsf and luf to represent the longest successor factor and the longest unbordered factor, respectively.

Problem Definition. The Longest Unbordered Factor Array problem can be defined formally as follows.

Longest Unbordered Factor Array
Input: A word w of length n.
Output: An array $\mathsf{LUF}[1\,..\,n]$ such that $\mathsf{LUF}[i]$ is the length of the maximal unbordered factor starting at position i in w, for all $1 \le i \le n$.

[1] The modified algorithm also computes some starting position $j > i$ for each factor $w[i\,..\,i+|\mathsf{LSF}_\ell[i]|-1]$, $1 \le i \le n$. Each such factor corresponds to the lowest common ancestor of the two terminal nodes in the suffix tree of w representing the suffixes $w[i\,..\,n]$ and $w[j\,..\,n]$; this ancestor can be located in constant time after linear-time preprocessing [1]. A linear-time preprocessing of the suffix tree also allows for constant-time computation of the rightmost starting position of each such factor.

▶ **Example 4.** Consider $w = \texttt{aabbabaabbaababbabab}$, then the longest unbordered factor array is as follows. (Observe that w is unbordered thus $\mu(w) = |w| = 20$.)

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$w[i]$	a	a	b	b	a	b	a	a	b	b	a	a	b	a	b	b	a	b	a	b
LUF[i]	20	3	12	9	12	3	14	3	11	3	10	5	2	3	5	2	2	2	2	1

3 Combinatorial Tools

The core of our algorithm exploits the unique unbordered-decomposition of all suffixes of w in order to compute the length of the maximal (longest) unbordered prefix of each such suffix. Let the unbordered-decomposition of $w[i \mathinner{.\,.} n]$ be $u_k \cdots u_1$ as in Proposition 2. Then $\mathsf{LUF}[i] = |u_k|$. In order to compute the unbordered-decomposition for all the suffixes *efficiently*, the algorithm uses the repetitive structure of w characterised by the longest successor factor arrays.

Basis of the algorithm. Abstractly, it is easy to observe that for a given position, if the length of the longest successor factor is zero (no factor starting at this position repeats afterwards) then the suffix starting at that position is necessarily unbordered. On the other hand, if the length of the longest successor factor is smaller than the length of the unbordered factor at the reference (the position of the the last occurrence of the longest successor factor) then the ending positions of the longest unbordered factors at this position and that at its reference will coincide; these two cases are formalised in Lemmas 5 and 6 below. The remaining case is not straightforward and its handling accounts for the bulk of the algorithm.

▶ **Lemma 5.** *If* $\mathsf{LSF}_\ell[i] = 0$ *then* $\mathsf{LUF}[i] = n - i + 1$*, for* $1 \le i \le n$.

▶ **Lemma 6.** *If* $\mathsf{LSF}_r[i] = j$ *and* $\mathsf{LSF}_\ell[i] < \mathsf{LUF}[j]$ *then* $\mathsf{LUF}[i] = j + \mathsf{LUF}[j] - i$*, for* $1 \le i \le n$.

Proof. Let $k = j + \mathsf{LUF}[j] - 1$. We first show that $w[i \mathinner{.\,.} k]$ is unbordered. Assume that $w[i \mathinner{.\,.} k]$ is bordered and let β be the length of one of its borders ($\beta < \mathsf{LSF}_\ell[i]$ as $\mathsf{LSF}_r[i] = j$). This implies that $w[i \mathinner{.\,.} i+\beta-1] = w[k-\beta+1 \mathinner{.\,.} k]$. Since $w[i \mathinner{.\,.} i+\mathsf{LSF}_\ell[i]-1] = w[j \mathinner{.\,.} j+\mathsf{LSF}_\ell[i]-1]$, we get $w[j \mathinner{.\,.} j+\beta-1] = w[k-\beta+1 \mathinner{.\,.} k]$ (i.e., $w[j \mathinner{.\,.} k]$ is bordered) which is a contradiction. Moreover, $w[k+1 \mathinner{.\,.} n]$ can be factorised into prefixes of $w[j \mathinner{.\,.} k]$ (by definition of LUF); every such prefix is also a proper prefix of $w[i \mathinner{.\,.} i+\mathsf{LSF}_\ell[i]-1]$ which will make every factor $w[i \mathinner{.\,.} k'], k < k' \le n$, to be bordered. This completes the proof. ◀

We introduce the notion of a *hook* to handle finding the unbordered-decomposition of suffixes $w[i \mathinner{.\,.} n]$ for the remaining case (i.e., when $\mathsf{LSF}_\ell[i] \ge \mathsf{LUF}[\mathsf{LSF}_r[i]]$).

▶ **Definition 7** (Hook)**.** Consider a position j in a length-n word w. Its *hook* $\mathcal{H}_j$ is the smallest position q such that $w[q \mathinner{.\,.} j-1]$ can be decomposed into unbordered prefixes of $w[j \mathinner{.\,.} n]$.

The following observation provides a greedy construction of this decomposition.

▶ **Observation 8.** *The decomposition of a word v into unbordered prefixes of another word u is unique. This decomposition can be constructed by iteratively trimming the shortest prefix of u which occurs as a suffix of the decomposed word.*

Moreover, the decomposability into unbordered prefixes of u is hereditary in a certain sense:

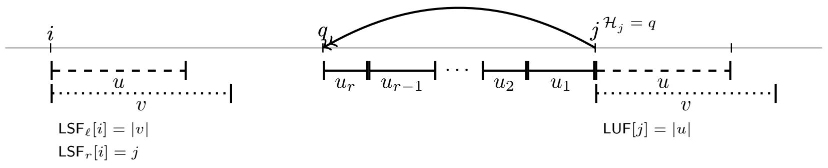

Figure 1 Case a ($i < q$): The unbordered-decomposition of $w[i\,.\,.\,n]$ consists of $w[i\,.\,.\,q-1]$ as the longest unbordered prefix, followed by a sequence of unbordered prefixes of u, including u itself at position j. Therefore, $\mathsf{LUF}[i] = q - i$.

▶ **Observation 9.** *If a word v can be decomposed into unbordered prefixes of u, then every prefix of v also admits such a decomposition. Formally, if $v = u_r \cdot u_{r-1} \cdot \ldots \cdot u_2 \cdot u_1$ such that each u_i, $r \ge i \ge 1$, is an unbordered prefix of u then any prefix $v[1\,.\,.\,k]$ can be uniquely decomposed as $v[1\,.\,.\,k] = u_r \cdot u_{r-1} \cdots u_{i-1} \cdot u'_p \cdot u'_{p-1} \cdots u'_1$, where k falls in u_i and each u'_i, $p \ge i \ge 1$, is an unbordered prefix of u; simply, the decomposition preceding u_i will be retained by the prefix.*

▶ **Example 10.** Consider w = `aabbabaabbaababbabab` as in Example 4. Observe that $\mathcal{H}_{18} = 13$: the factor $w[13\,.\,.\,17] =$ `ba` · `b` · `ba` can be decomposed into unbordered prefixes of $w[18\,.\,.\,20] =$ `bab`. Moreover, no prefix of $w[18\,.\,.\,20]$ matches a suffix of $w[1\,.\,.\,12] = \cdots$`aa`.

The hook $\mathcal{H}_j$ has its utility when j is a reference as shown in the following lemma.

▶ **Lemma 11.** *Consider a position i such that $\mathsf{LSF}_\ell[i] \ge \mathsf{LUF}[j]$, where $j = \mathsf{LSF}_r[i]$. Then*

$$\mathsf{LUF}[i] = \begin{cases} \mathcal{H}_j - i & \text{if } i < \mathcal{H}_j, \\ \mathsf{LUF}[j] & \text{otherwise.} \end{cases}$$

Proof. Let $u = w[j\,.\,.\,j + \mathsf{LUF}[j] - 1]$, $v = w[i\,.\,.\,i + \mathsf{LSF}_\ell[i] - 1]$, and $q = \mathcal{H}_j$. Observe that u occurs at position i and that $w[q\,.\,.\,n]$ can be decomposed into unbordered prefixes of u.

Case a: $i < q$. We shall prove that $w[i\,.\,.\,q-1]$ is the longest unbordered prefix of $w[i\,.\,.\,n]$; see Figure 1. By Observation 9, any longer factor $w[i\,.\,.\,k]$, $q \le k \le n$ has a suffix $w[q\,.\,.\,k]$ composed of unbordered prefixes of u. Thus, $w[i\,.\,.\,k]$ must be bordered, because u is its prefix. To conclude, for a proof by contradiction suppose that $w[i\,.\,.\,q-1]$ has a border v'. Note that $|v'| \le \mathsf{LSF}_\ell[i]$, so v' is a prefix of v. Hence, it occurs both as a suffix of $w[1\,.\,.\,q-1]$ and a prefix of $w[j\,.\,.\,n]$, which contradicts the greedy construction of $q = \mathcal{H}_j$ (Observation 8).

Case b: $i \ge q$. The decomposition of $w[q\,.\,.\,n]$ into unbordered prefixes of u yields a decomposition of $w[i\,.\,.\,n]$ into unbordered prefixes of u, starting with u. This is the unbordered-decomposition of $w[i\,.\,.\,n]$ (see Proposition 2), which yields $\mathsf{LUF}[i] = |u| = \mathsf{LUF}[j]$. ◀

4 Algorithm

The algorithm operates in two phases: a preprocessing phase followed by the main computation phase. The preprocessing phase accomplishes the following: Firstly, compute the longest successor factor array LSF_ℓ together with LSF_r array. If $\mathsf{LSF}_r[i] = j$ then we say *i refers to j* and mark j in a boolean array ($\mathsf{IsReference}$) as a reference.

In the main phase, the algorithm computes the lengths of the longest unbordered factors for all positions in w. Moreover, it determines $\mathsf{HOOK}[j] = \mathcal{H}_j$ for each *potential reference*, i.e., each position j such that $j = \mathsf{LSF}_r[i]$ and $\mathsf{LSF}_\ell[i] \ge \mathsf{LUF}[j]$ for some $i < j$; see Lemma 11.

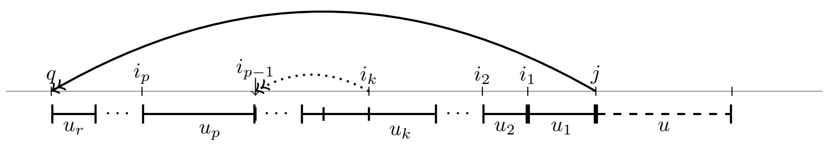

Figure 2 A chain of consecutive shortest prefixes of $w[j\,..\,n]$ starting at positions $i_1 > i_2 > \cdots > i_r = q$. No prefix of $w[j\,..\,n]$ is a suffix of $w[1\,..\,q-1]$, so the hook value of position j is $\mathcal{H}_j = q$. Meanwhile, $\mathsf{HOOK}[i_k]$ is set to i_{p-1} in order to avoid iterating through $i_{k+1}, \ldots, i_{p-1}$ again.

Positions are processed from right to left (in decreasing order) so that if i refers to j, then $\mathsf{LUF}[j]$ (and $\mathsf{HOOK}[j]$, if necessary) has already been computed before i is considered. For each position i, the value of $\mathsf{LUF}[i]$ is determined as follows:

1. If $\mathsf{LSF}_\ell[i] = 0$, then $\mathsf{LUF}[i] = n - i + 1$.
2. Otherwise
   a. If $\mathsf{LSF}_\ell[i] < \mathsf{LUF}[j]$, then $\mathsf{LUF}[i] = j + \mathsf{LUF}[j] - i$.
   b. If $\mathsf{LSF}_\ell[i] \geq \mathsf{LUF}[j]$ and $i \geq \mathsf{HOOK}[j]$, then $\mathsf{LUF}[i] = \mathsf{LUF}[j]$.
   c. If $\mathsf{LSF}_{\ell|}[i] \geq \mathsf{LUF}[j]$ and $i < \mathsf{HOOK}[j]$, then $\mathsf{LUF}[i] = \mathsf{HOOK}[j] - i$.

If i is a potential reference, then $\mathsf{HOOK}[i]$ is also computed, as described in Section 4.1. It is evident that the computational phase of the algorithm fundamentally reduces to finding the hooks for potential references; for brevity, the term reference will mean a potential reference hereafter.

4.1 Finding Hook (FindHook Function)

Main idea. When FindHook is called on a reference j, it must return $\mathcal{H}_j$. A simple greedy approach follows directly from Observation 8; see also Figure 2. Initially, the factor $w[1\,..\,j-1]$ is considered and the shortest suffix of $w[1\,..\,j-1]$ which is a prefix of $w[j\,..\,n]$ is computed. Then this suffix, denoted $u_1 = w[i_1\,..\,j-1]$, is truncated (chopped) from the considered factor $w[1\,..\,j-1]$; the next factor considered will be $w[1\,..\,i_1-1]$. In general, we iteratively compute and truncate the shortest prefixes of $w[j\,..\,n]$ from the right end of the considered factor; shortening the length of the considered factor in each iteration and terminating as soon as no prefix of $w[j\,..\,n]$ can be found. If the considered factor at termination is $w[1\,..\,q-1]$, position q is returned by the function as $\mathcal{H}_j$.

The factors $w[q\,..\,j-1]$ considered by successive calls of FindHook function may overlap. Moreover, the same *chains* of consecutive unbordered prefixes may be computed several times throughout the algorithm. To expedite the chain computation in the subsequent calls of FindHook on another reference j' ($j' < j$), we can *recycle* some of the computations done for j by shifting the value $\mathsf{HOOK}[\cdot]$ of each such index (at which a prefix was cut for j) leftwards (towards its final value). Consider the starting position i_k at which u_k was cut (i.e., $u_k = w[i_k\,..\,i_{k-1}-1]$ is the shortest unbordered prefix of $w[j\,..\,n]$ computed at i_{k-1}). Let i_p be the first position considered after i_k such that $|u_p| > |u_k|$. In this case, every factor $u_{k+1}, \ldots, u_{p-1}$ is a prefix of u_k; see Figure 2. Therefore, $w[i_{p-1}\,..\,i_k-1]$ can be decomposed into prefixes of u_k (and of $w[i_k\,..\,n]$). Consequently, we set $\mathsf{HOOK}[i_k] = i_{p-1}$ so that the next time a prefix of length greater than or equal to $|u_k|$ is cut at i_k, we do not have to repeat truncating the prefixes $u_{k+1}, \ldots, u_{p-1}$ and we may start directly from position i_{p-1}.

In order to express the intermediate values in the HOOK table, we generalize the notion of $\mathcal{H}_j$: for a position j and a length ℓ, we define $\mathcal{H}_j^\ell$ as the smallest position q such that $w[q\,..\,j-1]$ can be decomposed into unbordered prefixes of $w[j\,..\,n]$ whose lengths do not exceed ℓ. Observe that $\mathcal{H}_j^0 = j$ and $\mathcal{H}_j^\ell = \mathcal{H}_j$ if $\ell \geq \mathsf{LUF}[j]$.

Implementation. For each position i_k, we set $\mathsf{HOOK}[i_k] = \mathcal{H}_{i_k}^{|u_k|}$, equal to i_{p-1} in the case considered above. Computing these values for all indices i_k can be efficiently realised using a stack. Every starting position i_p, at which u_p is cut, is pushed onto the stack as a (length, position) pair $(|u_p|, i_p)$. Before pushing, every element $(|u_k|, i_k)$ such that $|u_k| < |u_p|$ is popped and the hook value of index i_k is updated ($\mathsf{HOOK}[i_k] = \mathcal{H}_{i_k}^{|u_k|} = i_{p-1} = i_p + |u_p|$).

Analysis. Throughout the algorithm, each unbordered prefix u_p at position i_p is computed just once by the FindHook function. Nevertheless, a longer[2] unbordered prefix u'_p may be computed at i_p again when FindHook is called on reference j' (where $q < j' < j$).

In what follows, we introduce certain characteristics of the computed unbordered prefixes which aids in establishing the relationship between the stacks of various references. Let $\mathcal{S}_j$ be the set of positions pushed onto the stack during a call of FindHook on reference j.

▶ **Definition 12** (Twin Set)**.** A *twin set* of reference j for length ℓ, denoted by $\mathcal{T}_j^\ell$, is the set of all the positions $i \in \mathcal{S}_j$ which were pushed onto the stack paired with length ℓ in the call of FindHook on reference j (i.e., $\mathcal{T}_j^\ell = \{i \mid (\ell, i) \text{ was pushed onto the stack of } j\}$).

Note that a *unique* shortest unbordered prefix of $w[j \mathinner{..} \mathsf{LUF}[j]-1]$ occurs at each i belonging to the same twin set. However, as and when a longer prefix at i is cut (say ℓ') for another reference $j' < j$, i will be added to $\mathcal{T}_{j'}^{\ell'}$.

▶ Remark. $\mathcal{S}_j = \bigcup_{\ell=1}^{\mathsf{LUF}[j]} \mathcal{T}_j^\ell$.

Hereafter, a twin set will essentially imply a non-empty twin set.

▶ **Lemma 13.** *If j' and j are references such that $j' \in \mathcal{S}_j$, then $\mathcal{H}_j \le \mathcal{H}_{j'}$.*

Proof. Since $j' \in \mathcal{S}_j$, the suffix $w[j' \mathinner{..} n]$ (and, by Observation 9, its every prefix $w[j' \mathinner{..} k]$) can be decomposed into unbordered prefixes of $w[j \mathinner{..} n]$. Consequently, any decomposition into unbordered prefixes of $w[j' \mathinner{..} n]$ yields a decomposition into unbordered prefixes of $w[j \mathinner{..} n]$. In particular, $w[\mathcal{H}_{j'} \mathinner{..} n]$ admits such a decomposition, which implies $\mathcal{H}_j \le \mathcal{H}_{j'}$. ◀

If the stack $\mathcal{S}_j$ is the most recent stack containing a reference j', we say that j' is the *parent* of j. More formally, the ***parent*** of j' is defined as $\min\{j \mid j' \in \mathcal{S}_j\}$. If j' does not belong to any stack (and thus has no parent), we will call it a ***base reference***.

▶ **Lemma 14.** *If j and j' are two references such that j is the parent of j' and $j' \in \mathcal{T}_j^\ell$, then each position $i \in \mathcal{S}_{j'}$ satisfies the following properties:*

1. *$i \in \mathcal{T}_j^\ell$;*
2. *there exists $k \in \mathcal{T}_j^{\ell'}$, with $\ell' > \ell$, such that $(k + \ell' - i, i)$ is pushed onto the stack of j'.*

Proof. Let p be the value of $\mathsf{HOOK}[j']$ prior to the execution of FindHook(j'). Since $j' \in \mathcal{T}_j^\ell$, the earlier call FindHook(j) has set $\mathsf{HOOK}[j'] = \mathcal{H}_{j'}^\ell$. As j is the parent of j', no further call has updated $\mathsf{HOOK}[j']$. Thus, we conclude that $p = \mathcal{H}_{j'}^\ell$.

Consequently, the first pair pushed onto the stack of j' is $(|z|, i)$, where $z = w[i \mathinner{..} p-1]$ is the shortest suffix of $w[1 \mathinner{..} p-1]$ which also occurs as a prefix of $w[j' \mathinner{..} n]$ (see Figure 3). Moreover, observe that $|z| > \ell$ by the greedy construction of $\mathcal{H}_{j'}^\ell$.

[2] It will be easy to deduce after Lemma 14 that the length of the prefix cut (the next time) at the same position will be at least twice the length of the current prefix cut at it.

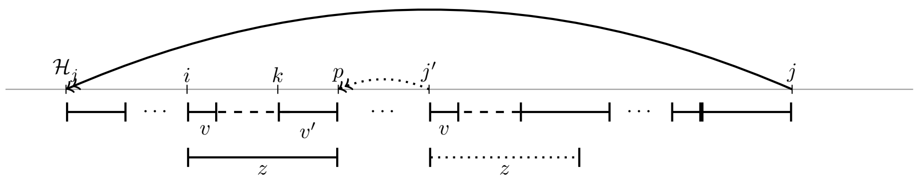

Figure 3 The pair $(|z|, i)$ is the first to be pushed onto the stack of j'. The factor z is unbordered, has v as a proper prefix and some v' as a proper suffix, where both v and v' are unbordered prefixes of $w[j\,..\,n]$ whose lengths ℓ and ℓ', respectively, satisfy $\ell < \ell'$.

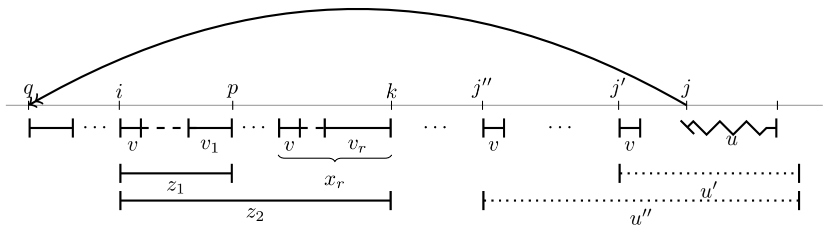

Figure 4 The pair $(|z_1|, i)$ and $(|z_2|, i)$ are pushed onto the stack of j' and j'' where i is a position common to both $\mathcal{S}_{j'}$ and $\mathcal{S}_{j''}$.

Recall that $j' \in \mathcal{T}_j^\ell$ implies that $w[j'\,..\,n]$ can be decomposed into unbordered prefixes of $w[j\,..\,n]$, with the first prefix of length ℓ, denoted $v = w[j'\,..\,j' + \ell - 1]$. With an occurrence at position j', the factor z also admits such a decomposition, still with the first prefix v (due to $|z| > |v|$). Additionally, note that $w[p\,..\,j' - 1]$ can be decomposed into unbordered prefixes of v. Concatenating the decompositions of $z = w[i\,..\,p - 1]$, $w[p\,..\,j' - 1]$, and $w[j'\,..\,n]$, we conclude that $w[i\,..\,n]$ can be decomposed into unbordered prefixes of $w[j\,..\,n]$ with the first prefix (in this unique decomposition) equal to v. Hence, $i \in \mathcal{S}_{j'}$ belongs to the same twin set as j'; i.e., it satisfies the first claim of the lemma.

Additionally, in the aforementioned decomposition of $w[i\,..\,n]$ consider the factor $v' = w[k\,..\,p - 1]$ which ends at position $p - 1$. By the greedy construction of $\mathcal{H}_{j'}^\ell$, its length $|v'|$ is strictly larger than ℓ, so $k \in \mathcal{T}_j^{\ell'}$ for $\ell' = |v'| > \ell$. Moreover, recall that $(|z|, i) = (k + \ell' - i, i)$ is pushed onto the stack of j'. Consequently, i also satisfies the second claim of the lemma.

A similar reasoning is valid for each i that will appear in $\mathcal{S}_{j'}$. ◀

▶ **Lemma 15.** *If j is the parent of two references $j'' < j'$, both of which belong to $\mathcal{T}_j^\ell$, then $\mathcal{S}_{j'} \cap \mathcal{S}_{j''} = \emptyset$.*

Proof. The proof is trivial if $\ell = \mathsf{LUF}[j]$. Let $\ell < \mathsf{LUF}[j]$, $u = w[j\,..\,j+\mathsf{LUF}[j]-1]$ and v be the shortest unbordered prefix of u cut at j' and j'' (i.e., $|v| = \ell$). Let $u' = w[j'\,..\,j' + \mathsf{LUF}[j'] - 1]$ and $u'' = w[j''\,..\,j'' + \mathsf{LUF}[j''] - 1]$. Here, the current call to the FindHook function has been made on the reference j''. Consider the largest position i such that it is common to the stacks of j' and j'' i.e. $i \in \mathcal{S}_{j'}$ and $i \in \mathcal{S}_{j''}$. Let the prefixes cut at i be $z_1 = w[i\,..\,p]$ and $z_2 = w[i\,..\,k]$. Observe that i being the largest position and $j' \neq j''$ ensure that $|z_1| \neq |z_2|$. Without loss of generality, let $|z_1| < |z_2|$ (examine Figure 4).

1. **j' cuts z_2 and j'' cuts z_1:** We proceed with the proof below by showing that there is a reference between j' and j that pushes j' onto its stack, thus contradicting the fact that j is the parent of j'.

Following Observation 9, $w[i\,..\,k]$ can be decomposed into unbordered prefixes of u'' with the first prefix being z_1 i.e. $z_2 = z_1 \cdot x_1 \cdot x_2 \cdot \ldots \cdot x_r$. Here, $|x_r| > |z_1|$ otherwise z_2 is bordered. Moreover, each x_i larger than v has corresponding position in $\mathcal{S}_{j''}$ and others (i.e. $|x_i| \le |v|$) are skipped because of $\mathsf{HOOK}[\cdot]$. Let x_s be the first of these $x_i, 1 \le i \le r$ such that $|x_s| > |z_1|$; the prefix $\tilde{z}_2 = z_1 \cdot \ldots \cdot x_s$ is unbordered. In the occurrence of z_2 at j', let j_0 be the position corresponding to x_s i.e. $j_0 = j' + |z_1 \cdots x_{s-1}|$.
Note that x_s, like every x_i and z_1, has v as proper prefix and some v_i as a proper suffix where v_i is an unbordered prefix of u longer than v (from Lemma 14). Therefore, $j_0 < j$ (x_s cannot start at j otherwise it would be bordered and x_s starting after j would contradict the assumption that j is the parent of j' as $w[j'\,..\,j_0]$ can be factorised into prefixes of x_s).
Now, we prove that j_0 is a (potential) reference. The fact that j' is a potential reference ensures that $\tilde{u} = w[j_0\,..\,j' + |u'| - 1]$ is a repeated factor. Moreover, $\tilde{u}$ contains the luf at j_0, say u_0, because u_0 is a factor (or suffix) of u' (since $w[j'\,..\,j_0 - 1]$ can be decomposed into prefixes of x_s); an implication is that $|\tilde{u}| \ge |u_0|$. Thus, j_0 is a reference if the last occurrence of $\tilde{u}$ is at j_0. For contradiction, assume that the factor $\tilde{u}$ has another occurrence at some position larger than j_0. This implies that there is another occurrence of u as u_0 contains u (the luf at any position which is in the stack of j, ends at or after $j + |u| - 1$). It is not possible as the last of the occurrences of u after j would cause j, j', j'' etc. to go in its stack and j would no longer be the parent of j' or j''.
Summing up, $j_0 < j$ is a reference with x_s as a prefix of u_0. If j is the parent of j_0 then j_0 would have pushed j' onto its stack, otherwise another reference j_{-1}, $j_0 < j_{-1} < j$ that pushed j_0 onto its stack would have pushed j' as well. In either case, j is not the parent of j' which is a contradiction.

2. ***j'* cuts *z_1* and *j''* cuts *z_2*:** Using the similar argument as in Case 1, we can prove that this case would lead to the conclusion that there is another reference between j'' and j that would push j'' onto its stack and hence contradicting that j is the parent of j''. ◀

4.2 Finding Shortest Border (FindBeta Function)

Given a reference j and a position q, function FindBeta returns the length β of the shortest prefix of $w[j\,..\,n]$ that is a suffix of $w[1\,..\,q-1]$, or $\beta = 0$ if there is no such prefix; note that the sought shortest prefix is necessarily unbordered.

To find this length, we use 'prefix-suffix queries' of [14, 13]. Such a query, given a positive integer d and two factors x and y of w, reports all prefixes of x of length between d and $2d$ that occur as suffixes of y. The lengths of sought prefixes are represented as an arithmetic progression, which makes it trivial to extract the smallest one. A single prefix-suffix query can be implemented in $\mathcal{O}(1)$ time after randomized preprocessing of w which takes $\mathcal{O}(n)$ time in expectation [14], or $\mathcal{O}(n \log n)$ time with high probability [13]. Additionally, replacing hash tables with deterministic dictionaries [16], yields an $\mathcal{O}(n \log n \log^2 \log n)$-time deterministic preprocessing.

To implement FindBeta, we set $x = [j\,..\,n]$, $y = [1\,..\,q-1]$ and we ask prefix-suffix queries for subsequent values $d = 1, 3, \ldots, 2^k - 1, \ldots$ until d exceeds $\min(|x|, |y|)$. Note that we can terminate the search as soon as a query reports a non-empty answer. Hence, the running time is $\mathcal{O}(1 + \log \beta)$ if the query is successful (i.e., $\beta \neq 0$) and $\mathcal{O}(\log n)$ otherwise.

Furthermore, we can expedite the successful calls to FindBeta if we already know that $\beta \notin \{1, \ldots, \ell\}$. In this case, we can start the search with $d = \ell + 1$. Specifically, if j is not a base reference and belongs to $\mathcal{T}^{\ell}_{j'}$ for some j', we can start from $d = 2\ell + 1$ because Lemma 14.2 guarantees that $\beta \ge \ell + \ell' > 2\ell$.

5 Analysis

Our algorithm computes the longest unbordered factor at each position i; position i is a start-reference or it refers to some other position. The correctness of the computed $\mathsf{LUF}[i]$ follows directly from Lemmas 5, 6 and 11.

The analysis of the algorithm running time necessitates probing of the total time consumed by FindHook and the time spent by FindBeta function which, in turn, can be measured in terms of the total size of the stacks of various references.

▶ **Lemma 16.** *The total size of all the stacks used throughout the algorithm is $\mathcal{O}(n \log n)$. Moreover, the total running time of the* FindBeta *function is $\mathcal{O}(n \log n)$.*

Proof. First, we shall prove that any position p belongs to $\mathcal{O}(\log n)$ stacks.

By Lemma 14.1, the stack of any reference is a subset of the stack of its parent. Moreover, by Lemmas 14.1 and 15, the stacks of references sharing the same parent are disjoint. A similar argument shows that the stacks of base references are disjoint.

Consequently, the references $j_1 > \ldots > j_s$ whose stacks $\mathcal{S}_{j_i}$ contain p form a chain with respect to the parent relation: j_1 is a base reference, and the parent of any subsequent j_i is j_{i-1}. Let us define $\ell_1, \ldots, \ell_s$ so that $p \in \mathcal{T}_{j_i}^{\ell_i}$. By Lemma 14.2, for each $1 \le i < s$, there exist k_i and $\ell'_i > \ell_i$ such that $k_i \in \mathcal{T}_{j_i}^{\ell'_i}$ and $\ell_{i+1} = k_i - p + \ell'_i \ge \ell_i + \ell'_i > 2\ell_i$. Due to $1 \le \ell_i \le n$, this yields $s \le 1 + \log n = \mathcal{O}(\log n)$, as claimed.

Next, let us analyse the successful calls $\beta = \text{FindBeta}(q, j)$ with $p = q - \beta$. Observe that after each such call, p is inserted to the stack $\mathcal{S}_j$ and to the twin set $\mathcal{T}_j^{\beta}$, i.e, $j = j_i$ and $\beta = \ell_i$ for some $1 \le i \le s$. Moreover, if $i > 1$, then $j_i \in \mathcal{T}_{j_{i-1}}^{\ell_{i-1}}$, which we are aware of while calling FindBeta. Hence, we can make use of the fact that $\ell_i \notin \{1, \ldots, 2\ell_{i-1}\}$ to find $\beta = \ell_i$ in time $\mathcal{O}(\log \frac{\ell_i}{\ell_{i-1}})$. For $i = 1$, the running time is $\mathcal{O}(1 + \log \ell_1)$. Hence, the overall running time of successful queries $\beta = \text{FindBeta}(q, j)$ with $p = q - \beta$ is $\mathcal{O}(1 + \log \ell_1 + \sum_{i=2}^{s} \log \frac{\ell_i}{\ell_{i-1}}) = \mathcal{O}(1 + \log \ell_s) = \mathcal{O}(\log n)$, which sums up to $\mathcal{O}(n \log n)$ across all positions p.

As far as the unsuccessful calls $0 = \text{FindBeta}(q, j)$ are concerned, we observe that each such call terminates the enclosing execution of FindHook. Hence, the number of such calls is bounded by n and their overall running time is clearly $\mathcal{O}(n \log n)$. ◀

▶ **Theorem 17.** *Given a word w of length n, our algorithm solves the* Longest Unbordered Factor Array *problem in $\mathcal{O}(n \log n)$ time with high probability. It can also be implemented deterministically in $\mathcal{O}(n \log n \log^2 \log n)$ time.*

Proof. Assuming an integer alphabet, the computation of LSF_ℓ and LSF_r arrays along with the constant time per position initialisation of the other arrays sum up the preprocessing stage to $\mathcal{O}(n)$ time. The running time required for the assignment of the luf for all positions is $\mathcal{O}(n)$. The time spent in construction of the data structure to answer prefix-suffix queries used in FindBeta function is $\mathcal{O}(n \log n)$ with high probability or $\mathcal{O}(n \log n \log^2 \log n)$ deterministic.

Additionally, the total running time of the FindHook function for all the references, being proportional to the aggregate size of all the stacks, can be deduced from Lemma 16. This has been shown to be $\mathcal{O}(n \log n)$ in the worst case, same as the total running time of FindBeta. The claimed bound on the overall running time follows. ◀

We can also show that the upper bound shown in Lemma 16 is in the worst case tight by designing an infinite family of words that exhibit the worst-case behaviour. We plan to include this construction in the full version of the paper.

6 Final Remark

Computing the longest unbordered factor in $o(n \log n)$ time for integer alphabets remains an open question.

References

1 Michael A. Bender and Martin Farach-Colton. The LCA Problem Revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

2 Patrick Hagge Cording and Mathias Bæk Tejs Knudsen. Maximal Unbordered Factors of Random Strings. In Shunsuke Inenaga, Kunihiko Sadakane, and Tetsuya Sakai, editors, *String Processing and Information Retrieval - 23rd International Symposium, SPIRE 2016, Beppu, Japan, October 18-20, 2016, Proceedings*, volume 9954 of *Lecture Notes in Computer Science*, pages 93–96, 2016. doi:10.1007/978-3-319-46049-9_9.

3 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

4 Maxime Crochemore and Lucian Ilie. Computing Longest Previous Factor in Linear Time and Applications. *Inf. Process. Lett.*, 106(2):75–80, 2008.

5 Maxime Crochemore, Lucian Ilie, Costas S. Iliopoulos, Marcin Kubica, Wojciech Rytter, and Tomasz Waleń. Computing the Longest Previous Factor. *Eur. J. Comb.*, 34(1):15–26, 2013.

6 Maxime Crochemore and Wojciech Rytter. *Jewels of stringology*. World Scientific, 2002. doi:10.1142/4838.

7 Jean-Pierre Duval. Relationship between the period of a finite word and the length of its unbordered segments. *Discrete Mathematics*, 40(1):31–44, 1982. doi:10.1016/0012-365X(82)90186-8.

8 Jean-Pierre Duval, Thierry Lecroq, and Arnaud Lefebvre. Linear computation of unbordered conjugate on unordered alphabet. *Theor. Comput. Sci.*, 522:77–84, 2014. doi:10.1016/j.tcs.2013.12.008.

9 Andrzej Ehrenfeucht and D. M. Silberger. Periodicity and unbordered segments of words. *Discrete Mathematics*, 26(2):101–109, 1979. doi:10.1016/0012-365X(79)90116-X.

10 Pawel Gawrychowski, Gregory Kucherov, Benjamin Sach, and Tatiana A. Starikovskaya. Computing the Longest Unbordered Substring. In Costas S. Iliopoulos, Simon J. Puglisi, and Emine Yilmaz, editors, *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings*, volume 9309 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2015. doi:10.1007/978-3-319-23826-5_24.

11 Stepan Holub and Dirk Nowotka. The Ehrenfeucht-Silberger problem. *J. Comb. Theory, Ser. A*, 119(3):668–682, 2012. doi:10.1016/j.jcta.2011.11.004.

12 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast Pattern Matching in Strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.

13 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Efficient Data Structures for the Factor Periodicity Problem. In Liliana Calderón-Benavides, Cristina N. González-Caro, Edgar Chávez, and Nivio Ziviani, editors, *String Processing and Information Retrieval - 19th International Symposium, SPIRE 2012, Cartagena de Indias, Colombia, October 21-25, 2012. Proceedings*, volume 7608 of *Lecture Notes in Computer Science*, pages 284–294. Springer, 2012. doi:10.1007/978-3-642-34109-0_30.

14 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Internal Pattern Matching Queries in a Text and Applications. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 532–551. SIAM, 2015. `doi:10.1137/1.9781611973730.36`.

15 Alexander Loptev, Gregory Kucherov, and Tatiana A. Starikovskaya. On Maximal Unbordered Factors. In Ferdinando Cicalese, Ely Porat, and Ugo Vaccaro, editors, *Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 - July 1, 2015, Proceedings*, volume 9133 of *Lecture Notes in Computer Science*, pages 343–354. Springer, 2015. `doi:10.1007/978-3-319-19929-0_29`.

16 Milan Ruzic. Constructing Efficient Dictionaries in Close to Sorting Time. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2008. `doi:10.1007/978-3-540-70575-8_8`.