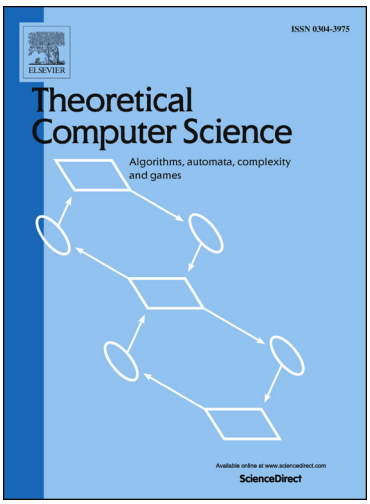
Accepted Manuscript



Reconstructing a string from its Lyndon arrays

Frantisek Franek, Jacqueline W. Daykin, Jan Holub, A.S.M. Sohidull Islam, W.F. Smyth

PII: S0304-3975(17)30330-4
DOI: http://dx.doi.org/10.1016/j.tcs.2017.04.008
Reference: TCS 11161

To appear in: *Theoretical Computer Science*

Received date: 13 June 2016
Revised date: 9 April 2017
Accepted date: 17 April 2017

Please cite this article in press as: F. Franek et al., Reconstructing a string from its Lyndon arrays, *Theoret. Comput. Sci.* (2017), http://dx.doi.org/10.1016/j.tcs.2017.04.008

Highlights

- An algorithm to determine whether or not a given array is a Lyndon array.
- Two algorithms to compute a string from a valid Lyndon array.
- An algorithm to compute a unique string from "rotated" Lyndon arrays.
- Plus theoretical background.

Reconstructing a String from its Lyndon Arrays*

Frantisek Franek**[1,4], Jacqueline W. Daykin[2,5], Jan Holub[3], A. S. M. Sohidull Islam[4], and W. F. Smyth**[1,4,5,6]

[1] Algorithms Research Group, Department of Computing & Software
McMaster University
{franek,smyth}@mcmaster.ca
[2] Department of Computer Science
Aberstwyth University (Mauritius Branch Campus)
jackie.daykin@gmail.com
[3] Department of Theoretical Computer Science
Czech Technical University in Prague
Jan.Holub@fit.cvut.cz
[4] School of Computational Science & Engineering
McMaster University
sohansayed@gmail.com
[5] Department of Informatics, King's College London
[6] School of Engineering & Information Technology
Murdoch University

Abstract. Given a string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ on an ordered alphabet Σ of size σ, the Lyndon array $\lambda = \lambda_{\boldsymbol{x}}[1..n]$ of $\boldsymbol{x}$ is an array of positive integers such that $\lambda[i],\ 1 \le i \le n$, is the length of the maximal Lyndon word over the ordering of Σ that begins at position i in $\boldsymbol{x}$. The Lyndon array has recently attracted considerable attention due to its pivotal role in establishing the long-standing conjecture that $\rho(n) < n$, where $\rho(n)$ is the maximum number of maximal periodicities (runs) in any string of length n. Here we first describe two linear-time algorithms that, given a valid Lyndon array λ, compute a corresponding string — one for an alphabet of size n, the other for a smaller alphabet. We go on to describe another linear-time algorithm that determines whether or not a given integer array is a Lyndon array of some string. Finally we show how σ Lyndon arrays $\lambda_\Sigma = \{\lambda_1 = \lambda, \lambda_2, \ldots, \lambda_\sigma\}$ corresponding to σ "rotations" of the alphabet can be used to determine uniquely the string $\boldsymbol{x}$ on Σ such that $\lambda_{\boldsymbol{x}} = \lambda$.

1 Introduction

We suppose throughout that $\boldsymbol{x}$ is a nonempty string. If $\boldsymbol{x} = \boldsymbol{uv}$ for some $\boldsymbol{u}$ and nonempty $\boldsymbol{v}$, then $\boldsymbol{vu}$ is said to be the $|\boldsymbol{u}|^{\text{th}}$ ***rotation*** of $\boldsymbol{x}$, written

* The authors express their thanks to two anonymous referees whose comments have materially improved the quality of this paper.
** Supported in part by grants from the Natural Sciences & Engineering Research Council (NSERC) of Canada.

$\boldsymbol{vu} = R_{|\boldsymbol{u}|}(x)$. A string $\boldsymbol{x}$ is said to be a ***repetition*** if $\boldsymbol{x} = \boldsymbol{u}^e$ for some nonempty string $\boldsymbol{u}$ and some integer $e \geq 2$; otherwise, $\boldsymbol{x}$ is said to be ***primitive***. A primitive string $\boldsymbol{x}$ that is lexicographically least among all its rotations $R_k(\boldsymbol{x}), k = 0, 1, \ldots, |\boldsymbol{x}|–1$, is said to be a ***Lyndon word*** [26]. As a consequence of their interesting properties, Lyndon words have been much studied: the existence of a unique factorization $\boldsymbol{x} = \boldsymbol{w_1 w_2 \cdots w_s}$ of a string into Lyndon words $\boldsymbol{w_1} \geq \boldsymbol{w_2} \geq \cdots \geq \boldsymbol{w_s}$ was demonstrated some 60 years ago [7] and a simple linear-time algorithm to compute the ***Lyndon factorization*** was proposed a quarter-century later [18].

In fact, Lyndon words are a special case of Unique Maximal Factorization Families (UMFFs), that over the last 15 years have also been studied extensively [16, 17, 13, 12]. When every factor $\boldsymbol{w_j}$, $1 \leq j \leq s$, of a (not necessarily Lyndon) factorization of $\boldsymbol{x}$ belongs to a specified set $\mathcal{W}$, we say that it is a ***factorization of $\boldsymbol{x}$ over $\mathcal{W}$***, denoted by $F_{\mathcal{W}}(\boldsymbol{x})$. Then a subset $\mathcal{W} \subseteq \Sigma^+$ is a ***factorization family*** (FF) if and only if for every nonempty string $\boldsymbol{x}$ on Σ there exists a factorization $F_{\mathcal{W}}(\boldsymbol{x})$. If $\mathcal{W}$ is an FF on an alphabet Σ, then $\mathcal{W}$ is said to be a ***unique maximal factorization family*** (UMFF) if and only if there exists a unique factorization $F_{\mathcal{W}}(\boldsymbol{x})$ for every string $\boldsymbol{x} \in \Sigma^+$. We expect that the results given here for Lyndon arrays can be generalized to UMFFs.

The ***Lyndon array*** $\boldsymbol{\lambda} = \boldsymbol{\lambda_x}[1..n]$ (equivalently, $L = L_{\boldsymbol{x}}[1..n]$) of a given $\boldsymbol{x} = \boldsymbol{x}[1..n]$ gives at each position i the length (equivalently, the end position) of the longest (or ***maximal***) Lyndon word starting at i. Thus $L_{\boldsymbol{x}}[i] = \lambda_{\boldsymbol{x}}[i]+i-1$. Apparently first introduced as "Lyndon bracketing" [25], the Lyndon array has recently become of interest because of the central role it plays in the surprising and simple proof [6] that the maximum number $\rho(n)$ of maximal periodicities (runs) in any string of length n satisfies $\rho(n) < n$. A recent paper [22] studies algorithms to compute $\lambda_{\boldsymbol{x}}$, exhibiting several that apparently execute in linear expected time, while conjecturing that there exists a worst-case linear-time algorithm to compute $\lambda_{\boldsymbol{x}}$ that is "elementary" — not a precise term, but we intend by it an algorithm that computes local features of a string while avoiding prior computation of global data structures such as the suffix array. Indeed, such an algorithm has recently been found [4, 5] as a first step in a two-step non-recursive linear-time suffix array construction algorithm. Here is an example of a Lyndon array, taken from [22]:

$$\begin{array}{rl} & \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \boldsymbol{x} = & \begin{array}{cccccccccc} a & b & a & a & b & a & b & a & a & b \end{array} \\ \boldsymbol{\lambda_x} = & \begin{array}{cccccccccc} 2 & 1 & 5 & 2 & 1 & 2 & 1 & 3 & 2 & 1 \end{array} \\ L_{\boldsymbol{x}} = & \begin{array}{cccccccccc} 2 & 2 & 7 & 5 & 5 & 7 & 7 & 10 & 10 & 10 \end{array} \end{array} \tag{1}$$

Since λ and L are arrays of positive integers, it is natural to ask under what conditions a given integer array is a Lyndon array. In Section 2 we give necessary and sufficient conditions that a given integer array L^* is a Lyndon array $L_{\boldsymbol{x}}$ of some string $\boldsymbol{x}$ on some alphabet Σ. We then describe linear-time algorithms that compute a string $\boldsymbol{x}$ corresponding to a given Lyndon array L^* — the problem of computing a lexicographically least such string on a minimum-size alphabet appears to be computationally difficult. Finally we describe a linear-time algorithm to determine whether or not a given integer array is a Lyndon array of some string.

In Section 3 we go on to establish a "reverse engineering" result for Lyndon arrays; that is, given certain Lyndon arrays $L_{\boldsymbol{x}}$ based on orderings of a given alphabet Σ of size σ, what can be said about the corresponding string $\boldsymbol{x}$? This kind of problem was first introduced in [21, 19] for the border array, then later considered for various common string data structures; for example, prefix tables [9, 3, 8], KMP arrays [20, 23, 24], cover arrays [10], and many others. Section 3 also presents an $\mathcal{O}(\sigma n)$-time algorithm to compute the unique string $\boldsymbol{x}$ determined by the Lyndon arrays computed for σ rotations of the alphabet. In Section 4 we discuss a variety of open problems arising.

2 When is L^* a Valid Lyndon Array of Some String?

We begin with a result from [22]:

Observation 1 *Suppose positions i, j in $\boldsymbol{x}[1..n]$ satisfy $1 \leq i < j \leq n$. Then either $L_{\boldsymbol{x}}[i] < j$ or $L_{\boldsymbol{x}}[i] \geq L_{\boldsymbol{x}}[j]$; that is, the vectors $(i, L_{\boldsymbol{x}}[i])$ and $(j, L_{\boldsymbol{x}}[j])$ are nonintersecting, and $L_{\boldsymbol{x}}[i] = j \Longrightarrow j = L_{\boldsymbol{x}}[j]$.*

(We remark that a generalization of this result follows also from the **xyz** Lemma of [16] which states that an FF $\mathcal{W}$ is an UMFF if and only if whenever $\boldsymbol{xy}, \boldsymbol{yz} \in \mathcal{W}$ for some nonempty $\boldsymbol{y}$, then $\boldsymbol{xyz} \in \mathcal{W}$. If we let $\boldsymbol{xy}$ be the Lyndon word at $\boldsymbol{x}[i..L_{\boldsymbol{x}}[i]]$ and $\boldsymbol{yz}$ the Lyndon word at $\boldsymbol{x}[j..L_{\boldsymbol{x}}[j]]$, it follows that the case $i < j < L_{\boldsymbol{x}}[i] < L_{\boldsymbol{x}}[j]$ cannot arise.)

Observation 1 tells us that a nonintersecting, or Monge-like, property necessarily holds for the arcs $(i, L[i])$ determined by the Lyndon array $L_{\boldsymbol{x}}$ of every string $\boldsymbol{x}$. To see that this property is also sufficient, consider an integer array $L^*[1..n]$ in which $i \leq L^*[i] \leq n$ for every $i \in 1..n$, and where either $L^*[i] < j$ or $L^*[i] \geq L^*[j]$ for every $1 \leq i < j \leq n$. Suppose an alphabet $\Sigma = \{\mu_1, \mu_2, \ldots, \mu_n\}$ is given, with $\mu_1 < \mu_2 < \cdots < \mu_n$. We now outline an algorithm (see Figure 1) that assigns the n letters of Σ to positions in $\boldsymbol{x}$ in such a way that $L_{\boldsymbol{x}} = L^*$.

procedure SimpleAssign $(L^*, n, \Sigma, \sigma; \boldsymbol{x})$
Radix sort pairs $(L^*[i], i),\ 1 \le i \le n$, in ascending
    order of i within descending order of $L^*[i]$
    to form sorted positions $I = I[1..n]$.
for $j \leftarrow 1$ **to** n **do**
    $\boldsymbol{x}[I[j]] \leftarrow j$

Fig. 1. Given a valid Lyndon array L^* and an ordered alphabet $\Sigma = \{1, 2, \ldots, n\}$, in $\mathcal{O}(n)$ time construct a string $\boldsymbol{x}$ on Σ whose Lyndon array is L^*.

This algorithm ensures that arcs $(i, L^*[i])$ are processed in descending order of $L^*[i]$ — specifically, so that for all $i_1, i_2, \ldots, i_m$ such that

$$L^*[i_1] = L^*[i_2] = \cdots = L^*[i_m],\ i_1 < i_2 < \cdots < i_m,$$

it follows that $\boldsymbol{x}[i_1] < \boldsymbol{x}[i_2] < \cdots < \boldsymbol{x}[i_m]$. Thus for each choice of $L^*[i]$, we ensure that $L^*[i] = L_{\boldsymbol{x}}[i]$. The descending order, together with the nonintersecting property, then guarantees that this identity holds for all i, and so the nonintersecting property is sufficient to ensure that L^* is the Lyndon array of some string — in particular $\boldsymbol{x}$. Thus:

Lemma 2 *Suppose that $L^*[1..n]$ is an integer array such that $1 \le L^*[i] \le n$ for all $i \in 1..n$. Then L^* is a Lyndon array $L_{\boldsymbol{x}}$ of some string $\boldsymbol{x}$ if and only if for all i, j such that $1 \le i < j \le n$, either $L^*[i] < j$ or $L^*[i] \ge L^*[j]$.*

For the example (1), Algorithm SimpleAssign yields the following:

$$\begin{array}{rcccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ L^* = & 2 & 2 & 7 & 5 & 5 & 7 & 7 & 10 & 10 & 10 \\ I = & 8 & 9 & 10 & 3 & 6 & 7 & 4 & 5 & 1 & 2 \\ \boldsymbol{x} = & 9 & 10 & 4 & 7 & 8 & 5 & 6 & 1 & 2 & 3 \end{array} \tag{2}$$

In an effort to construct a string $\boldsymbol{x}$ on a smaller alphabet, we employ a strategy (see Figure 2) that for each range of increasing values in I (that is, for each maximal Lyndon word of length at least two):

- chooses an initial letter one greater than the initial letter in the immediately following maximal Lyndon word;
- assigns the same letter to consecutive positions at the beginning of the current maximal Lyndon word — but excluding the final position;
- thereafter increments the letter by one at each successive position.

For example, in (2), after selecting $\boldsymbol{x}[8..10] = 112$, we choose

$$\boldsymbol{x}[3] = 2,\ \boldsymbol{x}[6] = 3,\ \boldsymbol{x}[7] = 4,$$

corresponding to $I[4..6] = 367$ and ensuring that $L_{\boldsymbol{x}}[5] \leq 5$. Then, for $I[7..8] = 45$, we choose

$$\boldsymbol{x}[4] = 4,\ \boldsymbol{x}[5] = 5,$$

finally yielding $\boldsymbol{x} = 3424534112$, on an alphabet of size 5 rather than 10, but still far from the minimum of 2 ($\boldsymbol{x} = 1211212112$). Clearly Algorithm BetterAssign also executes in $\mathcal{O}(n)$ time; it yields the same worst-case result as SimpleAssign (when $I = n, n-1, \ldots, 1$), but otherwise finds a string $\boldsymbol{x}$ on a smaller alphabet.

procedure BetterAssign $(L^*, n, \Sigma, \sigma; \boldsymbol{x})$
Compute $I[1..n]$ as in SimpleAssign
$I[n+1] \leftarrow 0; h \leftarrow 1; i \leftarrow 1$
while $i \leq n$ **do**
▷ *Assign letters to range of increasing values from* $I[i]$.
  repeat
  ▷ *Consecutive positions at start of range are identical.*
    $\boldsymbol{x}[I[i]] \leftarrow h;\ i \leftarrow i+1$
  until $I[i+1] \neq I[i]+1$
  if $I[i+1] < I[i]$ **then**
  ▷ *End position in range must be incremented.*
    $\boldsymbol{x}[I[i]] \leftarrow h+1;\ i \leftarrow i+1$
  else
    while $I[i+1] > I[i]$ **do**
    ▷ *Elsewhere in range every position is incremented.*
      $h \leftarrow h+1;\ \boldsymbol{x}[I[i]] \leftarrow h;\ i \leftarrow i+1$
▷ *Reset* h *depending on the next range in* $\boldsymbol{x}$.
  if $i \leq n$ **then**
  $j \leftarrow i$
  while $I[j+1] > I[j]$ **do** $j \leftarrow j+1$
    $h \leftarrow \boldsymbol{x}[I[j]+1]+1$

Fig. 2. Construct a string $\boldsymbol{x}$ on a subset of $\Sigma = \{1, 2, \ldots, n\}$ with Lyndon array L^*.

We know of no approach other than brute force (trial and error) to the computation of $\boldsymbol{x}$ on a minimum alphabet consistent with L^*. Hence

```
function CheckLyndon (L*, n)
STACK ← ∅
for i ← 1 to n do
    if L*[i] < i or L*[i] > n return (FALSE)
    if L*[i] > i then
        end ← 0
        while STACK ≠ ∅ and end < L*[i] do
            end ← peek(STACK)
            if end < i then pop(STACK)
            elsif end < L*[i] then return (FALSE)
        push(STACK, L*[i])
return (TRUE)
```

Fig. 3. Determine whether (TRUE) or not (FALSE) a given integer array L^* is a Lyndon array of some string.

Problem 3 *Given a valid Lyndon array L^*, what is the complexity of the problem of constructing a string $\boldsymbol{x}$ on a minimum alphabet consistent with L^*?*

We turn now to the problem of determining whether or not a given integer array L^* is valid; that is, whether or not it is a Lyndon array of some string. To solve this problem we introduce Algorithm CheckLyndon (see Figure 3), based on Lemma 2. It processes the segments $(i, L^*[i])$ in ascending order of position i and places the "*end*" of each nontrivial segment on the stack. Before doing so, it checks to see if any previous *end* lies within the current segment: if so, L^* cannot be a Lyndon array. If not, then either the previous entry ended before the current range and so can be deleted from the stack, or else it includes the current range and so must be kept in the stack to be tested against later segments. Note that entries in the stack have all been tested against *preceding* segments. We claim therefore that CheckLyndon is correct.

To see that the algorithm executes in linear time, observe that segment i is either wholly contained in a preceding segment, so that access to the stack is terminated, or else the current stack entry is deleted. Thus the total time requirement of the **while** loop is $\mathcal{O}(n)$. We have:

Lemma 4 *Algorithm CheckLyndon correctly determines in $\mathcal{O}(n)$ time whether or not a given integer array $L^*[1..n]$ is a Lyndon array.*

3 Reconstructing a String from its Lyndon Arrays

Suppose an alphabet $\Sigma = \{\ell_1, \ell_2, \ldots, \ell_\sigma\}$, $\sigma \geq 2$, is given with initial global order R_1: $\ell_1 < \ell_2 < \cdots < \ell_\sigma$. For $j = 2, 3, \ldots, \sigma$, the j^{th} rotation R_j of R_1 is the order $\ell_j < \ell_{j+1} < \cdots < \ell_\sigma < \ell_1 < \cdots < \ell_{j-1}$. Thus ℓ_j is the least letter, and for $j > 1$ ℓ_{j-1} is the largest, in the rotation R_j. The collection of σ rotations is denoted by R_Σ.

In this section we deal with the problem of identifying a unique string on alphabet Σ corresponding to R_Σ. We begin with an observation from [22], that we can write $\boldsymbol{x}$ in the form $\boldsymbol{x_1 x_2 \cdots x_m}$, where for each $r \in 1..m$, $|\boldsymbol{x_r}| = len_r$ and

$$\boldsymbol{x_r}[1] \leq \boldsymbol{x_r}[2] \leq \cdots \leq \boldsymbol{x_r}[len_r], \tag{3}$$

while for $1 \leq r < m$,

$$\boldsymbol{x_r}[len_r] > \boldsymbol{x_{r+1}}[1]. \tag{4}$$

We call $\boldsymbol{x_r}$ a ***range*** in $\boldsymbol{x}$, and we identify a position j in range $\boldsymbol{x_r}$, $1 \leq j \leq len_r$, with its equivalent position i in $\boldsymbol{x}$ by writing $i = S_{r,j} = \sum_{r'=1}^{r-1} len_{r'} + j$. Then, again from [22], we have the following:

Observation 5 *Let $i = S_{r,j}$ be a position in $\boldsymbol{x}$ that corresponds to position j in range $\boldsymbol{x_r}$.*

(*a*) *If $\boldsymbol{x_r}[j] = \boldsymbol{x_r}[len_r]$, then $L_{\boldsymbol{x}}[i] = i$.*
(*b*) *Otherwise, $L_{\boldsymbol{x}}[i] = i'$, where i' is the final position in some range $\boldsymbol{x_{r'}}$, $r' \geq r$; that is, $i' = \sum_{s=1}^{r'} len_s$.*

Based on these remarks, for the special case $\sigma = 2$, we can now prove:

Lemma 6 *Let $L_{\boldsymbol{x}}$ be the Lyndon array of a string $\boldsymbol{x}[1..n]$ on $\Sigma = \{a, b\}$, $a < b$. Then, provided that $\lambda_{\boldsymbol{x}} \neq 1^n$, $\boldsymbol{x}$ is determined uniquely by $L_{\boldsymbol{x}}$.*

Proof. First observe that $\lambda_{\boldsymbol{x}} = 1^n$ if and only if $\boldsymbol{x} = b^m a^{n-m}$ for some $m \in 0..n$ — thus in this case the corresponding $L_{\boldsymbol{x}} = 12 \cdots n$ corresponds to $n+1$ choices for $\boldsymbol{x}$. Otherwise, let n' be the smallest index such that for every $i \in n'..n$, there exists no $j < i$ such that $L_{\boldsymbol{x}}[j] = i$. If there is no such n', then $L_{\boldsymbol{x}}[j] = n$ for some $j < n$; in this case, set $n' = n+1$. Then for every $i \in n'..n$, $L_{\boldsymbol{x}}[i] = i$, which, since $\boldsymbol{x} \neq b^n$ by hypothesis, implies that $\boldsymbol{x}[i] = a$; that is, $\boldsymbol{x}[n'..n] = a^{n-n'+1}$. Since, again by hypothesis, $\boldsymbol{x} \neq a^n$, it follows that $n' > 2$ and $\boldsymbol{x}[n'-1] = b$. More generally, by Observation 5, for every $i < n'$, $\boldsymbol{x}[i] = b$ if and only if $L_{\boldsymbol{x}}[i] = i$. Therefore, for all other i, $\boldsymbol{x}[i] = a$. Thus $\boldsymbol{x}$ is uniquely determined by $L_{\boldsymbol{x}}$, as required. □

Suppose then that $\sigma \geq 3$. In this case, corresponding to R_Σ, we assume that Lyndon arrays $\lambda_\Sigma = \{\lambda_1, \lambda_2, \ldots, \lambda_\sigma\}$ are given, where λ_j, $1 \leq j \leq \sigma$, is based on rotation R_j of the alphabet and determined by some (unknown) string $\boldsymbol{x} = \boldsymbol{x}[1..n]$. Then for $j \in 1..\sigma$, $i \in 1..n$, $\lambda_j[i]$ is the length of the maximal Lyndon word at position i in $\boldsymbol{x}$ based on rotation R_j of the alphabet. We say that position i in $\boldsymbol{x}$, $1 \leq i \leq n$, is ***covered*** by λ_j if and only if there exists a position $i' < i$ such that $i' + \lambda_j[i'] > i$ (alternatively, $L_j[i'] \geq i$).

We now prove the following result, enabling us to uniquely determine $\boldsymbol{x}$ from R_Σ:

Theorem 7 *Suppose that $\boldsymbol{x}$ is a string on an alphabet Σ of size $\sigma \geq 3$, whose Lyndon arrays λ_Σ are given based on rotations R_Σ. Let*

$$\lambda^+[i] = max(\lambda_1[i], \lambda_2[i], \ldots, \lambda_\sigma[i]),\ 1 \leq i \leq n,$$

and let $P[i]$ be the nonempty ascending sequence $\{j_1, j_2, \ldots, j_k\}$ of indices j that specify all the rotations R_j for which $\lambda^+[i] = \lambda_j[i]$; that is, that yield the maximum Lyndon word at position i. Then

1. $|P[i]| = 1 \implies \boldsymbol{x}[i] = \ell_{j_1}$.
2. $1 < |P[i]| < \sigma \implies \boldsymbol{x}[i] = \ell_{j_h}$, *where j_h is the unique (leftmost) entry in $P[i]$ such that $j_{(h+1) \bmod \sigma} \notin P[i]$.*
3. $|P[i]| = \sigma \iff \lambda^+[i] = 1$, *and $\boldsymbol{x}[i..n] = \ell_{j_h}^{n-i+1}$, where i is the least integer such that $\lambda^+[i] = 1$ and j_h is the unique entry in $P[i]$ such that $\boldsymbol{x}[i]$ is* not *covered by λ_{j_h}.*

We remark that, since the assignments to positions in $\boldsymbol{x}$ made here under Cases 1–3 are unique, therefore $\boldsymbol{x}$ must be the only string on Σ that satisfies the constraints given by λ_Σ. In Figure 4 the various cases of Theorem 7 are illustrated:

$$\begin{array}{rll}
i = & 1\ 2\ 3\ 4\ 5 & \\
\lambda_1 = & 1\ \underline{4}\ 3\ 2\ 1 & (a < b < c) \\
\lambda_2 = & \underline{2}\ 1\ \underline{3}\ \underline{2}\ 1 & (b < c < a) \\
\lambda_3 = & 1\ 3\ 1\ 1\ \underline{1} & (c < a < b) \\
\lambda^+ = & 2\ 4\ 3\ 2\ 1 & \\
k = & 1\ 1\ 2\ 2\ 3 & \\
\boldsymbol{x} = & b\ a\ b\ b\ c &
\end{array}$$

Fig. 4. Lyndon arrays based on rotated orders for $\sigma = 3$.

- in columns $i = 1, 2$, the respective maximum values $\lambda_2[1] = 2,\ \lambda_1[2] = 4$ occur only once, so that Case 1 applies, and $\boldsymbol{x}[1] = b,\ \boldsymbol{x}[2] = a$;
- in column $i = 3$, we find $k = 2 < \sigma$, so Case 2 applies, and since $2 \in P[3],\ 3 \notin P[3]$, we choose $\boldsymbol{x}[3] = \ell_2 = b$;
- similarly in column $i = 4$, Case 2 appies and again we choose $\boldsymbol{x}[4] = b$;
- of course in column $i = n = 5$, Case 3 applies, and we set $\boldsymbol{x}[5] = c$ because, while position 5 is covered by preceding entries in λ_1 and λ_2, it is not covered by any preceding entry in λ_3.

In order to prove Theorem 7, we first need the following:

Lemma 8 *Let $i \in 1..n$ be a position in $\boldsymbol{x}$ such that $\boldsymbol{x}[i] = \ell_j \in \Sigma$ for some $j \in 1..\sigma$, $\sigma \geq 3$. Then $\lambda^+[i] = \lambda_j[i] \geq \lambda_{j'}[i]$ for every $\ell_{j'} \in \Sigma$.*

Proof. Assume the contrary. Then there exists $j' \neq j$ ($\ell_{j'} \neq \ell_j$) such that $\lambda_{j'}[i] > \lambda_j[i]$. Suppose now that for some position h satisfying $i < h < i+\lambda_j[i]$ (in the range of the Lyndon word corresponding to R_j that begins at i), $\boldsymbol{x}[h] = \ell_*$, where ℓ_* is a letter such that $\ell_* < \ell_j$ in $R_{j'}$. But this implies that $\lambda_{j'}[i] < \lambda_j[i]$, contradicting our original assumption. Thus every letter ℓ that occurs in the range $\boldsymbol{x}[i..i+\lambda_j-1]$ must satisfy $\ell \geq \ell_j$ in both R_j and $R_{j'}$. Since the same condition holds for both rotations, therefore $\lambda_{j'}[i]$ can be at most equal to $\lambda_j[i]$. □

If now we suppose, as in Case 1 of Theorem 7, that there exists a single j_1 such that $\lambda_{j_1}[i]$ is maximum, then it follows immediately from Lemma 8 that, for every rotation R_t, $t \in 1..\sigma$, except $t = j_1$, $\boldsymbol{x}[i] \neq \ell_t$. Thus $\boldsymbol{x}[i] = \ell_{j_1}$, establishing Case 1.

The next result gives us a basis for establishing Case 2 by providing a simple characterization of the entries in $P[i]$ when $1 < |P[i]| < \sigma$:

Lemma 9 *Suppose that j_1 is the least value and $j_2 > j_1$ the greatest value (with $j_2-j_1 < \sigma-1$) such that for some $i \in 1..n$, $\lambda^+[i] = \lambda_{j_1}[i] = \lambda_{j_2}[i]$. Then $P[i]$ contains exactly one of*

$$P_1 = j_1, j_1+1, \ldots, j_2-1,$$
$$P_2 = j_2, j_2+1, \ldots, \sigma, 1, 2, \ldots, j_1-1,$$

where we auppose that the sequence $1, 2, \ldots, j_1-1$ is empty if $j_1 = 1$.

Proof. By hypothesis $\boldsymbol{x}^* = \boldsymbol{x}[i..i+\lambda^+[i]-1]$ is a Lyndon word in both orders R_{j_1} and R_{j_2}. Therefore every letter in $\boldsymbol{x}^*$ must be greater than or equal to $\boldsymbol{x}[i]$ in *both* orders

$$\begin{aligned} J_1 &= \{j_1 < j_1{+}1 < \cdots < j_2{-}1\}, \\ J_2 &= \{j_2 < j_2{+}1 < \cdots < \sigma < 1 < 2 \cdots < j_1{-}1\}, \end{aligned} \tag{5}$$

where again we must take account of the special case $j_1 = 1$. We can write $R_{j_1} \equiv J_1\{j_2{-}1 < j_2\}J_2$ and $R_{j_2} \equiv J_2\{j_1{-}1 < j_1\}J_1$. Now observe that if $\boldsymbol{x}^*$ contains letters from *both* J_1 and J_2, there must be at least one letter in one of the two orderings that is less than $\boldsymbol{x}[i]$, and so $\boldsymbol{x}^*$ cannot be a Lyndon word in one of R_{j_1}, R_{j_2}. Thus $\boldsymbol{x}^*$ contains letters from exactly one of P_1, P_2, as required. □

In the context of Lemma 9, consider a maximal sequence of entries

$$j', j'{+}1, \ldots, j'{+}t,\ t > 0, \tag{6}$$

in $\lambda^+[i]$, where $(j'{+}t{+}1) \bmod \sigma \notin \lambda^+[i]$ — as in Lemma 9, we suppose that the sequence is circular, so that 1 follows σ. Recall that $R_{j'+1}$ is the rotation of $R_{j'}$ that turns the least letter $\ell_{j'}$ of rotation $R_{j'}$ into the greatest letter of rotation $R_{j'+1}$. Thus the occurrence of $j'{+}1$ in the sequence $\lambda^+[i]$ ensures that the letter $\ell_{j'}$ cannot be the first letter of the Lyndon array at position i — if it were, then in $R_{j'+1}$, we could have only $\lambda_{j'}[i] = 1$, certainly not maximum. It follows that only the final letter $\ell_{j'+t}$ in the sequence (6) can be the first letter of the Lyndon array at i, because it is the only letter that is not rotated. Noting that in both of the two possible orders given in (5) $j'{+}t$ will be the *leftmost* occurrence in $P[i]$, we thus establish Case 2 of Theorem 7.

In order to deal with Case 3, we first need:

Lemma 10 $|P[i]| = \sigma \iff \lambda^+[i] = 1$.

Proof. Suppose $|P[i]| = \sigma$. Since every letter in Σ occurs on the right in some rotation R_j of the alphabet, and so is maximum, it follows that $\lambda_j[i] = 1$ for some j. But since every such j yields a maximum $\lambda_j[i]$, it follows that $\lambda^+[i] = 1$. Conversely, if $\lambda^+[i] = 1$, then every rotation R_j yields $\lambda_j[i] = 1$, so that $|P[i]| = \sigma$. □

Now consider $i = n$, and note that for every rotation R_j, $\lambda_j[n] = 1$, so that $\lambda^+[n] = 1$. Note also that in this case $\boldsymbol{x}[n]$ will be covered by $\lambda_j[i]$ for every position $i < n$ that marks the rightmost occurrence of ℓ_j in $\boldsymbol{x}$

— except for j such that $\ell_j = \boldsymbol{x}[i] = \boldsymbol{x}[n]$. Thus the letter ℓ_j satisfies exactly one of two conditions:

- either ℓ_j does not previously occur in $\boldsymbol{x}$; or
- under rotation R_j, at any position $i < n$ such that $\boldsymbol{x}[i] = \ell_j$, $\lambda_j[i]$ does not cover $\boldsymbol{x}[n]$.

Hence Case 3 provides the basis for assigning $\boldsymbol{x}[n]$. To complete the proof of Theorem 7, we need one more result:

Lemma 11 *If $|P[i]| = \sigma$ for $i < n$, then $|P[i']| = \sigma$ for every $i' \in i..n$.*

Proof. We know $\boldsymbol{x}[i] = \ell_j$, a minimum letter under rotation R_j, and from Lemma 10 we know that $\lambda^+[i] = 1$. This is possible only if the same minimum letter occurs also at positions $i{+}1, i{+}2, \ldots, n$, as required. □

Therefore Case 3 identifies strings $\boldsymbol{x}$ with suffix ℓ^{n-i+1} for some i and some ℓ.

Theorem 7 justifies Algorithm ConstructString, shown in Figure 5, that in $\mathcal{O}(\sigma n)$ time constructs the unique string $\boldsymbol{x}$ on a given ordered alphabet Σ, based on σ rotations of a given Lyndon array λ.

4 Final Remarks

In this paper we have started to analyze the relationship between a string and its Lyndon arrays corresponding to cyclic orderings of the underlying alphabet. Many problem areas remain:

1. As indicated by Problem 3, we know of no efficient algorithm to compute a string on a minimum alphabet consistent with a given valid Lyndon array L^*.
2. Similarly, as we have seen, it appears to be difficult to reconstruct a string exactly from Lyndon arrays. In Section 3 we have presented an algorithm to reconstruct a string $\boldsymbol{x}$ on an alphabet of size σ given Lyndon arrays of $\boldsymbol{x}$ based on σ rotations of the alphabet. It appears that indeed in the worst case σ such rotations are required, as shown by the example in Figure 6.

   If in Figure 6 only rotations I–III are considered, then the selection $\boldsymbol{x}[2] = c$ rather than d would be made; if rotations II–IV were used, no determination could be made for $\boldsymbol{x}[1]$; and if rotations I, III, IV were used, column $i = 3$ would become an instance of Case 2(b), and

```
procedure ConstructString (λ, σ, n; x)
i ← 1; cover ← σ^n
while i < n do
    maxlen ← 0    — Length of maximum λ[1..σ, i]
    for j ← 1 to σ do
▷ Treat λ as a two-dimensional array; find final
▷ letter jmax & frequency freqmax for maxlen.
        if λ[j, i] > maxlen then
            maxlen ← λ[j, i]; jmax ← j; freqmax ← 1
        elsif λ[j, i] = maxlen then
            jmax ← j; freqmax ← freqmax+1
▷ Recompute maximum range covered by letter j.
        cover[j] ← max(cover[j], j+λ[j, i])
    if freqmax < σ then
▷ Cases 1 and 2 — Lemmas 8 & 9.
        x[i] ← jmax; i ← i+1
    else
▷ Case 3 — Lemma 10.
        j' ← 1
▷ Find the letter that yields no cover of position i.
        while j' ≤ σ and i < cover[j'] do j' ← j'+1
        x[i] ← j'; i ← i+1
▷ In accordance with Lemma 11, extend to position n.
        while i ≤ n do x[i] ← j'; i ← i+1
```

Fig. 5. Constructing a string from rotated Lyndon arrays.

$\boldsymbol{x}[2] = b$ could not be selected. Thus it appears that, on an alphabet of size $\sigma > 2$, the Lyndon array does not strongly determine the underlying string.

$$
\begin{array}{rl}
i = & 1\ 2\ 3\ 4 \\
\boldsymbol{x} = & \mathrm{a\ d\ b\ c} \\
\lambda_1 = & 4\ 1\ 2\ 1 \quad (a < b < c < d)\ \mathrm{I} \\
\lambda_2 = & 1\ 1\ 2\ 1 \quad (b < c < d < a)\ \mathrm{II} \\
\lambda_3 = & 1\ 2\ 1\ 1 \quad (c < d < a < b)\ \mathrm{III} \\
\lambda_4 = & 1\ 3\ 2\ 1 \quad (d < a < b < c)\ \mathrm{IV}
\end{array}
$$

Fig. 6. $\boldsymbol{x} = adbc$ and its Lyndon arrays: consideration of fewer than four rotations of the alphabet may not allow $\boldsymbol{x}$ to be reconstructed.

3. Thus it would be of interest to determine minimal criteria for the reconstruction of $\boldsymbol{x}$ — the least number of rotations or the size of the smallest alphabet. Is there any hope of reconstructing a string based on fewer than σ permutations of the alphabet?
4. The study of UMFFs has led to the idea of V-words, analogous structures to Lyndon words, derived from a global non-lexicographic ordering of strings called V-order [11, 16, 14, 15]. Also, linear-time algorithms for computing Lyndon border and Lyndon suffix arrays have recently been proposed [2]. There is scope to eztend the results of this paper to the UMFF based on V-order [1], then further to UMFFs in their full generality.

References

1. A. Alatabbi, J. W. Daykin, J. Kärkkäinen, M. S. Rahman, and W. F. Smyth. V-Order: new combinatorial properties & a simple comparison algorithm. *Discrete Appl. Math.*, 215:41–46, 2016.
2. A. Alatabbi, J. W. Daykin, and M. S. Rahman. Linear algorithms for computing the Lyndon border array and the Lyndon suffix array. 2015. *arXiv:1506.06983*.
3. A. Alatabbi, M. S. Rahman, and W. F. Smyth. Inferring an indeterminate string from a prefix graph. *Journal of Discrete Algorithms*, 32:6–13, 2015.
4. U. Baier. Linear-time suffix sorting — a new approach for suffix array construction. M.Sc. Thesis, University of Ulm, 2015.
5. U. Baier. Linear-time suffix sorting — a new approach for suffix array construction. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, volume 54 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:12, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
6. H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta. The "runs" theorem. 2014. *arXiv:1406.0263v6*.
7. K. T. Chen, R. H. Fox, and R. C. Lyndon. Free differential calculus. iv. the quotient groups of the lower central series. *Annals of Mathematics*, 68(1):81–95, 1958.
8. M. Christodoulakis, P. J. Ryan, W. F. Smyth, and S. Wang. Indeterminate strings, prefix arrays and undirected graphs. *Theoretical Comput. Sci.*, 600:34–48, 2015.
9. J. Clément, M. Crochemore, and G. Rindone. Reverse engineering prefix tables. In *Proc. 26th Symp. on Theoretical Aspects of Computer Science*, pages 289–300, 2009.
10. M. Crochemore, C. S. Iliopoulos, S. P. Pissis, and G. Tischler. Cover array string reconstruction. In *Symp. on Combinatorial Pattern Matching*, volume 6129 of *Lecture Notes in Comput. Sci.*, pages 251–259. Springer, 2010.
11. T.-N. Danh and D. E. Daykin. The structure of V–order for integer vectors. *Congressus Numerantium*, 113:43–53, 1996.
12. D. E. Daykin, J. W. Daykin, C. S. Iliopoulos, and W. F. Smyth. Generic algorithms for factoring strings. In *Proc. Memorial Symp. for Rudolf Ahlswede, H. Aydinian, F. Cicalese & C. Deppe, (eds.)*, volume 7777 of *Lecture Notes in Comput. Sci.*, pages 402–418. Springer-Verlag, 2013.

13. D. E. Daykin, J. W. Daykin, and W. F. Smyth. Combinatorics of unique maximal factorization families (UMFFs). *Fund. Inform. 97–3, Special Issue on Stringology, R. Janicki, S. J. Puglisi & M. S. Rahman (eds.)*, pages 295–309, 2009.
14. D. E. Daykin, J. W. Daykin, and W. F. Smyth. String comparison and Lyndon–like factorization using V–order in linear time. In *22nd Annual Symp. on Combinatorial Pattern Matching*, volume 6661 of *Lecture Notes in Computer Science*, pages 65–76. Springer-Verlag, 2011.
15. D. E. Daykin, J. W. Daykin, and W. F. Smyth. A linear partitioning algorithm for hybrid Lyndons using V–order. *Theoret. Comput. Sci.*, 483:149–161, 2013.
16. D.E. Daykin and J.W. Daykin. Lyndon–like and V–order factorizations of strings. *J. Discrete Algorithms*, 1(3–4):357–365, 2003.
17. D.E. Daykin and J.W. Daykin. Properties and construction of unique maximal factorization families for strings. *Internat. J. Found. Comput. Sci.*, 19(4):1073–1084, 2008.
18. J.-P. Duval. Factorizing words over an ordered alphabet. *J. Algorithms*, 4:363–381, 1983.
19. J.-P. Duval, T. Lecroq, and A. Lefebvre. Border array on bounded alphabet. *Journal of Automata, Languages and Combinatorics*, 10(1):51–60, 2005.
20. J.-P. Duval, T. Lecroq, and A. Lefebvre. Efficient validation and construction of Knuth-Morris-Pratt arrays. In *Proc. Conference in Honour of Donald E. Knuth*, 2007.
21. F. Franek, S. Gao, W. Lu, P. J. Ryan, W. F. Smyth, Y. Sun, and L. Yang. Verifying a border array in linear time. *J. Combinatorial Math. & Combinatorial Computing*, 42:223–236, 2002.
22. F. Franek, A. S. M. S. Islam, M. S. Rahman, and W. F. Smyth. Algorithms to compute the Lyndon array. In Jan Holub and Jan Žďárek, editors, *Proceedings of the Prague Stringology Conference 2016*, pages 172–184, Czech Technical University in Prague, Czech Republic, 2016.
23. P. Gawrychowski, A. Jeż, and L. Jeż. Validating the Knuth-Morris-Pratt failure function, fast and on-line. In *Proc. 5th Annual Computer Science Symposium in Russia*, volume 6072 of *Lecture Notes in Comput. Sci.*, pages 132–143. Springer-Verlag, 2010.
24. P. Gawrychowski, A. Jeż, and L. Jeż. Validating the Knuth-Morris-Pratt failure function, fast and on-line. *Theory of Computing Systems*, 54:337–372, 2014.
25. J. Sawada and F. Ruskey. Generating Lyndon brackets: an addendum to "Fast algorithms to generate necklaces, unlabeled necklaces amd irreducible polynomials over GF(2)". *J. Algorithms*, 46:21–26, 2003.
26. B. Smyth. *Computing Patterns in Strings*. Pearson/Addison–Wesley, 2003.