A linear partitioning algorithm for Hybrid Lyndons using V-order

Daykin, David E.; Daykin, Jacqueline W.; Smyth, William F.

Published in:
Theoretical Computer Science

DOI:
10.1016/j.tcs.2012.02.001

Publication date:
2013

Citation for published version (APA):
Daykin, D. E., Daykin, J. W., & Smyth, W. F. (2013). A linear partitioning algorithm for Hybrid Lyndons using V-order. *Theoretical Computer Science*, *483*, 149-161. https://doi.org/10.1016/j.tcs.2012.02.001

Theoretical Computer Science 483 (2013) 149–161

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

A linear partitioning algorithm for Hybrid Lyndons using V-order☆

David E. Daykin [a], Jacqueline W. Daykin [b,c,*], W.F. Smyth [c,d]

[a] Department of Mathematics, University of Reading, UK

[b] Department of Computer Science, Royal Holloway, University of London, UK

[c] Department of Informatics, King's College, University of London, UK

[d] Algorithms Research Group, Department of Computing & Software, McMaster University, Hamilton ON L8S 4K1, Canada

ARTICLE INFO

Keywords:
Algorithm
Alphabet
Circ-UMFF
Concatenate
Factor
Hybrid Lyndon
Lexicographic order
Lyndon
Maximal
RAM
String
Total order
UMFF
V-order
Word
V-word

ABSTRACT

In this paper we extend previous work on unique maximal factorization families (UMFFs) and a total (but non-lexicographic) ordering of strings called V-order. We present new combinatorial results for V-order, in particular concatenation under V-order. We propose linear-time RAM algorithms for string comparison in V-order and for Lyndon-like factorization of a string into V-words. This asymptotic efficiency thus matches that of the corresponding algorithms for lexicographical order. Finally, we introduce Hybrid Lyndon words as a generalization of standard Lyndon words, and hence propose extensions of factorization algorithms to other forms of order.

1. Introduction

This paper explores a generalization of Lyndon words known as circ-UMFFs; that is, families of strings which permit the unique maximal factorization of any given string [9,10,12]. For over half a century, both the combinatorics and algorithmics of Lyndon words have been studied extensively [4,16,19,22]. Lyndon words have been applied in tackling a surprisingly wide range of problems: the Burrows–Wheeler Transform and data compression [3,17], musicology [1], bioinformatics [15], cryptanalysis [20], string combinatorics [12,18,22], and free Lie algebras [21]. We therefore study circ-UMFFs as generalizations of Lyndon words in order to extend the range of applications.

A ***circ-UMFF*** $\mathcal{W}$ (see below, Definitions 2.2 and 2.5) is an infinite collection of strings on some alphabet Σ, such that every string in Σ^* can be expressed as a unique concatenation of maximum-length elements of $\mathcal{W}$ – what we call a ***max factorization*** – with the additional requirement that no two elements of $\mathcal{W}$ can be rotations of each other. It turns out [9,10] that for a circ-UMFF $\mathcal{W}$, the strings $\boldsymbol{w_i}$ of $\mathcal{W}$ satisfy an ***UMFF-order*** $<_{\mathcal{W}}$, a total order over all the elements of $\mathcal{W}$ such that every max factorization $\boldsymbol{w_1 w_2 \cdots w_k}$ of a string $\boldsymbol{x}$ over $\mathcal{W}$ satisfies

$$\boldsymbol{w_1} \geq_{\mathcal{W}} \boldsymbol{w_2} \geq_{\mathcal{W}} \cdots \geq_{\mathcal{W}} \boldsymbol{w_k}. \tag{1}$$

☆ A preliminary version of these results appeared in Daykin et al. (2011) [13]. The authors acknowledge the contribution of two anonymous referees whose suggestions and ideas have materially improved this paper. The work of the third author was supported in part by the Natural Sciences & Engineering Research Council of Canada.

* Corresponding author at: Department of Informatics, King's College, University of London, UK.
E-mail addresses: J.Daykin@cs.rhul.ac.uk, jackie.daykin@kcl.ac.uk (J.W. Daykin), smyth@mcmaster.ca (W.F. Smyth).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.02.001

Thus, by virtue of its defining properties, a circ-UMFF satisfies, in terms of UMFF-order, an analogue or generalization of the Lyndon factorization theorem [4], which guarantees that the Lyndon words $\mathcal{L}$, in terms of lexicographic order (***lexorder***), provide a unique maximal factorization (1) of every string $\boldsymbol{x}$. In other words, $\mathcal{L}$ is an *example* of a circ-UMFF $\mathcal{W}$, suggesting that other orderings of the strings $\boldsymbol{x} \in \Sigma^+$ may yield alternative circ-UMFFs distinct from $\mathcal{L}$. Various alternative orderings were subsequently introduced in [9]:

- binary circ-UMFFs including B-words from the Block total order and T-words from the Type total order;
- the V-word circ-UMFF derived from V-order on an arbitrary alphabet [6,8].

The latter is our focus here.

In this paper we first extend combinatorial insights into V-order and the associated circ-UMFF factors, V-words. For instance, we show that, unlike lexorder, any subsequence of a string precedes the string in V-order. Surprisingly, although V-order is not immediately intuitive, it gives rise to V-words which exhibit similar structure to Lyndon words. One similarity is that when factoring into Lyndon words, a lesser letter than the first denotes the start of a new factor, while with V-words this occurs with a larger letter. So, interestingly, V-words are analogues of Lyndon words, defined using a simple extension of lexorder, ***lex-extension***, a combination of lexorder and V-order. These various insights lead to efficient new RAM algorithms for comparing strings in V-order and for factoring them into V-words. See [2] for a representation of rhythms in Central African music in terms of such a factorization.

More generally, when comparing strings lexicographically, any secondary ordering method can be applied to prescribed substrings. Based on the general form of lex-extension ordering, we define the concept of ***Hybrid Lyndons*** and show that V-words are an instance of this new class of Lyndon words. Then, since V-words are defined using lex-extension, an associated factorization algorithm naturally arises from the modification of existing Lyndon partitioning algorithms [16].

The Lyndon factorization for both RAM and PRAM algorithms has linear complexity [7,11,14,16,22], achievable with or without the use of recursion. We compare substrings $\boldsymbol{u}$, $\boldsymbol{v}$ nonrecursively in V-order, also achieving linear time by means of a method based on identification of the longest common suffix of the substrings $\boldsymbol{u}$, $\boldsymbol{v}$ and implemented using a simple doubly-linked list.

The currently identified (and, as far as we know, non-exhaustive) classes of circ-UMFFs are type Flight Deck and type Acrobat [12]. Since both Lyndon words and V-words belong to the Flight Deck class, which requires that all letters in a factor are not less than the first in UMFF-order, this paper also advances knowledge of this class of circ-UMFFs. We compare and contrast results for concatenation of Lyndon versus V-word factors: concatenation of Lyndon words is always in the same order as the defining lexorder of those words [16], whereas V-words demonstrate a variety of orders; indeed these two types of factors even exhibit different UMFF-behavior for a pair of strings according to the ordering of the first letters in each string. Similarly to the co-Lyndon circ-UMFF [12], the V-word alphabet is in reverse order to that of the defining V-order.

We show that for a given string $\boldsymbol{x}$, the lexicographic factorization of $\boldsymbol{x}$ into Lyndon words is distinct from the V-order factorization of $\boldsymbol{x}$ into V-words. Hence we have different methods for string factorization. Once a string has been decomposed over a circ-UMFF, say the Lyndon words, the factors are then maximal and thus final. Given an alternative circ-UMFF, say V-words, the Lyndon factors can be re-factored allowing for deeper burrowing into the string, for example yielding a subsequence of V-word prefixes, suffixes or substrings of all the Lyndon factors. Thus, algorithms for additional circ-UMFFs may lead to computing further subsequences of interest.

Additionally, the existence of a variety of circ-UMFFs opens the way to new optimization problems, for instance minimizing/maximizing the number of factors in a string partition. Indeed, given linear factorization methods, it costs linear time overall to determine an optimal factorization.

The structure of this paper is as follows. We first review the circ-UMFF theory relevant to this work in Section 2, followed by a discussion of V-order and associated new combinatorial results in Section 3. An algorithm for fast V-order string comparison is described in Section 4, then applied in Section 5 to achieve linear-time Lyndon-like factorization using V-order.

Note that the terms ***string*** and ***word*** mean the same thing (see References), hence we use both here. Also, for any examples included which use either the Roman alphabet or the non-negative integers, the natural order will be assumed, that is $\{a < b < c < \cdots < z\}$ and $\{0 < 1 < 2 < \cdots\}$. All strings are written in mathbold: $\boldsymbol{x}$, $\boldsymbol{w}$, and so on.

2. Unique maximal factorization families (UMFFs)

We commence with a summary of relevant UMFF theory from the fuller version given in [12]. Note that we use standard terminology from automata theory and stringology (see definitions in [22]). For brevity we write ***lexorder*** for lexicographic order, and ***co-lexorder*** for complementary lexorder; then, if a string $\boldsymbol{x}$ is less than a string $\boldsymbol{y}$ in lexorder, we write $\boldsymbol{x} < \boldsymbol{y}$ and $\boldsymbol{y} > \boldsymbol{x}$.

A string $\boldsymbol{w}$ is called a ***factor*** of a string $\boldsymbol{x}[1..n]$ if and only if $\boldsymbol{w} = \boldsymbol{x}[i..j]$ for $1 \le i \le j \le n$. Note that a factor is necessarily nonempty. If $\boldsymbol{x} = \boldsymbol{w_1 w_2 \cdots w_k}$, $1 \le k \le n$, then $\boldsymbol{w_1 w_2 \cdots w_k}$ is said to be a ***factorization*** of $\boldsymbol{x}$; moreover, when every factor $\boldsymbol{w_j}$, $1 \le j \le k$, belongs to a specified set $\mathcal{W}$, this is a ***factorization of x over*** $\mathcal{W}$, denoted by $F_{\mathcal{W}}(\boldsymbol{x})$.

Definition 2.1. 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 $F_{\mathcal{W}}(\boldsymbol{x})$.

For some string $\boldsymbol{x}$ and some FF $\mathcal{W}$, suppose $\boldsymbol{x} = \boldsymbol{w_1 w_2 \cdots w_k}$, where $\boldsymbol{w_j} \in \mathcal{W}$ for every $1 \le j \le k$. For some $1 \le k' \le k$, write $\boldsymbol{x} = \boldsymbol{u w_{k'} v}$, where $\boldsymbol{u} = \boldsymbol{w_1 w_2 \cdots w_{k'-1}}$ (empty if $k' = 1$) and $\boldsymbol{v} = \boldsymbol{w_{k'+1} w_{k'+2} \cdots w_k}$ (empty if $k' = k$). Suppose that there does not exist a suffix $\boldsymbol{u'}$ of $\boldsymbol{u}$ nor a prefix $\boldsymbol{v'}$ of $\boldsymbol{v}$ such that $\boldsymbol{u' w_{k'} v'} \neq \boldsymbol{w_{k'}}$ and $\boldsymbol{u' w_{k'} v'} \in \mathcal{W}$; then $\boldsymbol{w_{k'}}$ is said to be a ***max factor*** of $\boldsymbol{x}$. If *every* factor $\boldsymbol{w_{k'}}$ is max, then the factorization $F_{\mathcal{W}}(\boldsymbol{x})$ is itself said to be ***max***. Observe that a max factorization must be unique: there exists no other max factorization of $\boldsymbol{x}$ that uses only elements of $\mathcal{W}$.

Definition 2.2. Let $\mathcal{W}$ be an FF on an alphabet Σ. Then $\mathcal{W}$ is a ***unique maximal factorization family*** (UMFF) if and only if there exists a max factorization $F_{\mathcal{W}}(\boldsymbol{x})$ for every string $\boldsymbol{x} \in \Sigma^+$.

Lemma 2.3 (*The* **xyz** *Lemma [9]*)**.** *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 $\boldsymbol{xyz} \in \mathcal{W}$.*

The next result establishes that a max factorization is indeed a partition of a string with no overlapping factors. It is also relevant to the parallel PRAM Lyndon algorithm in [14] where recursion is applied to refactor previously factored substrings.

Corollary 2.4 (*[12]*)**.** *Suppose $\boldsymbol{x} = \boldsymbol{u_1 u_2 \cdots u_m}$ and $\mathcal{W}$ is an UMFF, where for every $1 \le j \le m$, $\boldsymbol{u_j} \in \mathcal{W}$. Then we can write the max factorization $F_{\mathcal{W}}(\boldsymbol{x}) = \boldsymbol{w_1 w_2 \cdots w_k}$, where*

$$\boldsymbol{w_1} = \boldsymbol{u_{j_0+1} \cdots u_{j_1}}, \qquad \boldsymbol{w_2} = \boldsymbol{u_{j_1+1} \cdots u_{j_2}}, \ldots, \qquad \boldsymbol{w_k} = \boldsymbol{u_{j_{k-1}+1} \cdots u_{j_k}},$$

$0 = j_0 < j_1 < j_2 < \cdots < j_{k-1} < j_k = m$.

Our algorithms execute on circ-UMFFs, with an associated rule for concatenation. To define these ideas, we first give a few basic stringological definitions [22]:

- If for some string $\boldsymbol{x}$ we can write $\boldsymbol{x} = \boldsymbol{uv} = \boldsymbol{wu}$ for some nonempty $\boldsymbol{u}$, then we say that $\boldsymbol{x}$ has ***border*** $\boldsymbol{u}$; if no such $\boldsymbol{u}$ exists, then $\boldsymbol{x}$ is said to be ***border-free***.
- If we can write $\boldsymbol{x} = \boldsymbol{u}^k$ for some nonempty $\boldsymbol{u}$ and some integer $k > 1$, we say that $\boldsymbol{x}$ is a ***repetition***; otherwise, we say that $\boldsymbol{x}$ is ***primitive***.
- If a string $\boldsymbol{x} = \boldsymbol{uv}$, then $\boldsymbol{vu}$ is said to be a ***rotation*** (cyclic shift) of $\boldsymbol{x}$, specifically the $|\boldsymbol{u}|^{\text{th}}$ rotation $R_{|\boldsymbol{u}|}(\boldsymbol{x})$, where $|\boldsymbol{u}| \in \{0, \ldots, |\boldsymbol{x}|\}$. Note that $R_0(\boldsymbol{x}) = R_{|\boldsymbol{x}|}(\boldsymbol{x})$.

Definition 2.5. An UMFF $\mathcal{W}$ over Σ^+ is a ***circ-UMFF*** if and only if it contains exactly one rotation of every primitive string $\boldsymbol{x} \in \Sigma^+$.

The set of strings consisting of increasing integers is an example of an UMFF that is not a circ-UMFF (see [10] for further examples).

Definition 2.6. If a circ-UMFF $\mathcal{W}$ contains strings $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{uv}$, we write $\boldsymbol{u} <_{\mathcal{W}} \boldsymbol{v}$ (called the ***$\mathcal{W}$-order***).

The next theorem provides characterizations of circ-UMFFs that will be frequently used in the following sections:

Theorem 2.7 (*[10]*)**.** *Let $\mathcal{W}$ be a circ-UMFF.*

(1) *If $\boldsymbol{u} \in \mathcal{W}$ then $\boldsymbol{u}$ is border-free.*
(2) *If $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{W}$ and $\boldsymbol{u} \neq \boldsymbol{v}$ then $\boldsymbol{uv}$ is primitive.*
(3) *If $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{W}$ and $\boldsymbol{u} \neq \boldsymbol{v}$ then $\boldsymbol{uv} \in \mathcal{W}$ or $\boldsymbol{vu} \in \mathcal{W}$ (but not both).*
(4) *If $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{uv} \in \mathcal{W}$ then $\boldsymbol{u} <_{\mathcal{W}} \boldsymbol{v}$ and $<_{\mathcal{W}}$ is a total order of $\mathcal{W}$.*
(5) *If $\boldsymbol{w} \in \mathcal{W}$ and $|\boldsymbol{w}| \ge 2$ then there exist $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{W}$ with $\boldsymbol{w} = \boldsymbol{uv}$.*

An example of $\mathcal{W}$-order is lexorder which gives the Lyndon circ-UMFF, denoted here as $\mathcal{L}$. Recall [16] that for Lyndon words, the $\mathcal{W}$-order of the factors is just lexorder, the same as the order normally applied to the strings in Σ^*.

Suppose now that the elements of Σ^* are totally ordered under $<_{\mathcal{T}}$, say, and that $\mathcal{W}$ is a circ-UMFF. It may then happen that for distinct strings $\boldsymbol{u}$ and $\boldsymbol{v}$, $\boldsymbol{u} <_{\mathcal{T}} \boldsymbol{v}$ while $\boldsymbol{v} <_{\mathcal{W}} \boldsymbol{u}$ (see [12]). Thus if $<_{\mathcal{T}}$ is used to select a rotation as in Definition 2.5 (for example, lexorder for Lyndon words), then we may need to distinguish whether the generated $\mathcal{W}$-order is coincidental or otherwise to the order $<_{\mathcal{T}}$.

Definition 2.8. Suppose that the strings of Σ^* are ordered according to a total order $<_{\mathcal{T}}$, and that $\mathcal{W}$ is a circ-UMFF on Σ^*. If, for every $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{W}$ such that $\boldsymbol{u} <_{\mathcal{W}} \boldsymbol{v}$,

(1) it follows that $\boldsymbol{u} <_{\mathcal{T}} \boldsymbol{v}$, then we write $\mathcal{W} \equiv \mathcal{T}$;
(2) it follows that $\boldsymbol{v} <_{\mathcal{T}} \boldsymbol{u}$, then we write $\mathcal{W} \equiv \overline{\mathcal{T}}$.

A simple example of $\mathcal{W} \equiv \overline{\mathcal{T}}$ arises if we define $<_{\mathcal{T}}$ as lexorder while $\mathcal{W}$ is defined as ***co-lexorder*** (that is, lexorder of reversed strings [12]). Note that defining $\mathcal{W}$ as lexorder on some other ordering of Σ would yield $\mathcal{W} \not\equiv \mathcal{T}$ *and* $\mathcal{W} \not\equiv \overline{\mathcal{T}}$. See below, lexicographic extension, Definition 3.9 and Lemma 3.16.

The following result generalizes the Lyndon factorization theorem [4] to circ-UMFFs, and is relevant to the algorithm of Section 5 (*cf.* Corollary 2.4).

Theorem 2.9 (*[12]*). *Let $\mathcal{W}$ be a circ-UMFF and let $\boldsymbol{x} = \boldsymbol{u_1 u_2 \cdots u_m}$ with each $\boldsymbol{u_j} \in \mathcal{W}$. Then $F_{\mathcal{W}}(\boldsymbol{x}) = \boldsymbol{u_1 u_2 \cdots u_m}$ if and only if*

$$\boldsymbol{u_1} \geq_{\mathcal{W}} \boldsymbol{u_2} \geq_{\mathcal{W}} \cdots \geq_{\mathcal{W}} \boldsymbol{u_m}.$$

The factors in a circ-UMFF exhibit nested structural properties:

Lemma 2.10 (*[10]*). *Suppose that $\boldsymbol{w}$ is an element of a circ-UMFF $\mathcal{W}$. If $\boldsymbol{u_1}, \boldsymbol{u_2}, \ldots, \boldsymbol{u_{k_1}}$ are all the proper prefixes of $\boldsymbol{w}$ in increasing order of length that belong to $\mathcal{W}$, and if $\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_{k_2}}$ are all the proper suffixes of $\boldsymbol{w}$ in decreasing order of length that belong to $\mathcal{W}$, then*

$$\boldsymbol{u_1} <_{\mathcal{W}} \boldsymbol{u_2} <_{\mathcal{W}} \cdots <_{\mathcal{W}} \boldsymbol{u_{k_1}} <_{\mathcal{W}} \boldsymbol{w} <_{\mathcal{W}} \boldsymbol{v_1} <_{\mathcal{W}} \boldsymbol{v_2} <_{\mathcal{W}} \cdots <_{\mathcal{W}} \boldsymbol{v_{k_2}}.$$

On the other hand, consider the variation in ordering that may occur in circ-UMFFs:

Lemma 2.11 (*[12]*). *Let $\mathcal{W}$ be a circ-UMFF and suppose $\boldsymbol{xy}, \boldsymbol{yz} \in \mathcal{W}$ for nonempty $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ (hence $\boldsymbol{x} \neq \boldsymbol{z}$). Then $\boldsymbol{xyz} \in \mathcal{W}$, $\boldsymbol{xyyz} \in \mathcal{W}$, and*

(1) $\boldsymbol{xy} <_{\mathcal{W}} \boldsymbol{xyz} <_{\mathcal{W}} \boldsymbol{yz}$;
(2) $\boldsymbol{xy} <_{\mathcal{W}} \boldsymbol{xyyz} <_{\mathcal{W}} \boldsymbol{yz}$;
(3) *either* $\boldsymbol{xyyzxyz} \in \mathcal{W}$ *or* $\boldsymbol{xyzxyyz} \in \mathcal{W}$ *(but not both).*

The case $\boldsymbol{xyyz} <_{\mathcal{W}} \boldsymbol{xyz}$ of Lemma 2.11(3) occurs for the Lyndon circ-UMFF based on lexicographic ordering [12]. As we shall see at the end of Section 3, the other choice in Lemma 2.11(3), $\boldsymbol{xyz} <_{\mathcal{W}} \boldsymbol{xyyz}$, arises in the V-order circ-UMFF, in which moreover both choices actually occur.

In [12] we introduced Flight Deck and Acrobat circ-UMFFs, and in this paper we are working with the former:

Definition 2.12. A circ-UMFF $\mathcal{W}$ is said to be ***type Flight Deck*** if and only if $\boldsymbol{w}[1 \ldots n] \in \mathcal{W}$ implies that for every $1 \leq i \leq n$, $\boldsymbol{w}[1] \leq_{\mathcal{W}} \boldsymbol{w}[i]$.

3. Combinatorics of V-order

We begin with the definition for totally ordering a set of strings according to V-order, followed first by some preliminary lemmas and then by the definition of V-order factors called V-words. Hence in this section $\mathcal{W}$-order is denoted as $\mathcal{V}$-order. (See [8,9] for related results and discussion on V-order and V-words; see [6] for an order equivalent to V-order due to Strehl and Winkelmann.)

Let Σ be a totally ordered alphabet, and let $\boldsymbol{u} = u_1 u_2 \ldots u_n$ be a string over Σ. Define $h \in \{1, \ldots, n\}$ by $h = 1$ if $u_1 \leq u_2 \cdots \leq u_n$; otherwise, by the unique value such that $u_{h-1} > u_h \leq u_{h+1} \leq u_{h+2} \leq \cdots \leq u_n$. Let $\boldsymbol{u}^* = u_1 u_2 \ldots u_{h-1} u_{h+1} \ldots u_n$, where the star * indicates deletion of the "V" letter u_h. Write $\boldsymbol{u}^{s*}$ for $(\cdots(\boldsymbol{u}^*)^* \cdots)^*$ with $s \geq 0$ stars.[1] Let $\mu\boldsymbol{u} = max\{u_1, u_2, \ldots, u_n\}$, say $\mu\boldsymbol{u} = g$, and let $k = \nu\boldsymbol{u}$ be the number of occurrences of g in $\boldsymbol{u}$. Then the sequence $\boldsymbol{u}, \boldsymbol{u}^*, \boldsymbol{u}^{2*}, \ldots$ ends $g^k, \ldots, g^2, g^1, g^0 = \boldsymbol{\varepsilon}$. In the *star tree* each string $\boldsymbol{u}$ over Σ labels a vertex, and there is a directed edge from $\boldsymbol{u}$ to $\boldsymbol{u}^*$, with $\boldsymbol{\varepsilon}$ as the root.

Definition 3.1. We define V-order $\prec$ between distinct strings $\boldsymbol{u}, \boldsymbol{v}$. First $\boldsymbol{v} \prec \boldsymbol{u}$ if $\boldsymbol{v}$ is in the path $\boldsymbol{u}, \boldsymbol{u}^*, \boldsymbol{u}^{2*}, \ldots, \boldsymbol{\varepsilon}$. If $\boldsymbol{u}, \boldsymbol{v}$ are not in a path, there exist smallest s, t such that $\boldsymbol{u}^{(s+1)*} = \boldsymbol{v}^{(t+1)*}$. Put $\boldsymbol{c} = \boldsymbol{u}^{s*}$ and $\boldsymbol{d} = \boldsymbol{v}^{t*}$; then $\boldsymbol{c} \neq \boldsymbol{d}$ but $|\boldsymbol{c}| = |\boldsymbol{d}| = m$ say. Let j be the greatest i in $1 \leq i \leq m$ such that $\boldsymbol{c}[i] \neq \boldsymbol{d}[i]$. If $\boldsymbol{c}[j] < \boldsymbol{d}[j]$ in Σ then $\boldsymbol{u} \prec \boldsymbol{v}$. Clearly $\prec$ is a total order.

We will be ordering the rotations of a string in V-order $\prec$, illustrated by:

Example 3.2. In V-order, $a \prec b \prec ab \prec bbb \prec c \prec abc \prec abbc \prec cbc \prec bcc$. For the rotations of $abbc$, we have $cabb \prec bcab \prec bbca \prec abbc$.

In order to describe our combinatorial results we introduce the ***V-form*** of a string $\boldsymbol{x}$, a unique representation of $\boldsymbol{x}$ defined as

$$V_k(\boldsymbol{x}) = \boldsymbol{x} = \boldsymbol{x_0} g \boldsymbol{x_1} g \cdots \boldsymbol{x_{k-1}} g \boldsymbol{x_k}, \tag{2}$$

for possibly empty $\boldsymbol{x_i}$, $i = 0, 1, \ldots, k$, where g is the largest letter in $\boldsymbol{x}$. (Thus we suppose that g occurs exactly k times.) Observe that if g_1 is the second largest letter in $\boldsymbol{x}$, then V-form can be recursively (and independently) applied to each $\boldsymbol{x_i}$ that includes g_1 with frequency $k_1 > 0$:

$$V_{k_1}(\boldsymbol{x_i}) = \boldsymbol{x_{i,0}} g_1 \boldsymbol{x_{i,1}} \cdots \boldsymbol{x_{i,k_1-1}} g_1 \boldsymbol{x_{i,k_1}}.$$

[1] Note that this star operator, as defined in [8,9,13], is distinct from the Kleene star operator.

We now clarify how the sequence $\boldsymbol{u}, \boldsymbol{u}^*, \boldsymbol{u}^{2*}, \ldots$ ends $g^k, \ldots, g^2, g^1, g^0 = \boldsymbol{\varepsilon}$ in the star tree. Denote by $A^T(\boldsymbol{x})$ the set of ancestors of $\boldsymbol{x}$ in the star tree, and by $A_j(\boldsymbol{x})$ the jth ancestor, $0 \le j \le |\boldsymbol{x}|$ (the zeroth ancestor is $\boldsymbol{x}$ itself, the $|\boldsymbol{x}|$th ancestor is the empty string $\boldsymbol{\varepsilon}$).

Lemma 3.3. *If* $V_k(\boldsymbol{x}) = \boldsymbol{x_0} g \boldsymbol{x_1} g \cdots \boldsymbol{x_{k-1}} g \boldsymbol{x_k}$, *then*

$$A_{t_j}(\boldsymbol{x}) = \boldsymbol{x_0^T} g \boldsymbol{x_1^T} \cdots \boldsymbol{x_{k-1}^T} g \boldsymbol{x_k^T},$$

where $j \in 0..k$, $t_j = |\boldsymbol{x_j}| + |\boldsymbol{x_{j+1}}| + \cdots + |\boldsymbol{x_k}|$, *and*

$$\begin{aligned} \boldsymbol{x_h^T} &= \boldsymbol{x_h} \quad \text{for } 0 \le h < j, \\ &= \boldsymbol{\varepsilon} \quad \text{for } j \le h \le k. \end{aligned}$$

Thus $A_{t_0}(\boldsymbol{x}) = g^k$, *while for* $j \ge 1$, $A_{t_j}(\boldsymbol{x})$ *has suffix* g^{k-j+1} *and* $A_{t_{j-1}}(\boldsymbol{x}) = A_{|\boldsymbol{x_{j-1}}|}(A_{t_j}(\boldsymbol{x}))$.

Our interest is to uniquely factor a string by applying V-order rather than lexorder. The following lemma forms a basis for the design of the factorization algorithm in Section 5.

Lemma 3.4 ([9]). *Given distinct strings* $\boldsymbol{v}$ *and* $\boldsymbol{x}$ *with corresponding V-forms*

$$\boldsymbol{v} = \boldsymbol{v_0} g_{\boldsymbol{v}} \boldsymbol{v_1} g_{\boldsymbol{v}} \cdots \boldsymbol{v_{j-1}} g_{\boldsymbol{v}} \boldsymbol{v_j}, \qquad \boldsymbol{x} = \boldsymbol{x_0} g_{\boldsymbol{x}} \boldsymbol{x_1} g_{\boldsymbol{x}} \cdots \boldsymbol{x_{k-1}} g_{\boldsymbol{x}} \boldsymbol{x_k},$$

let $h \in 0..\max(j, k)$ *denote the least integer such that* $\boldsymbol{v_h} \ne \boldsymbol{x_h}$. *Then* $\boldsymbol{v} \prec \boldsymbol{x}$ *(respectively,* $\boldsymbol{v} \succ \boldsymbol{x}$*) if and only if one of the following conditions holds:*
(1) $\mu\boldsymbol{v} < \mu\boldsymbol{x}$ *(respectively,* $\mu\boldsymbol{v} > \mu\boldsymbol{x}$*);*
(2) $\mu\boldsymbol{v} = \mu\boldsymbol{x}$ *and* $j < k$, *that is* $\nu\boldsymbol{v} < \nu\boldsymbol{x}$ *(respectively,* $j > k$*);*
(3) $\mu\boldsymbol{v} = \mu\boldsymbol{x}$ *and* $j = k$ *and* $\boldsymbol{v_h} \prec \boldsymbol{x_h}$ *(respectively,* $\boldsymbol{v_h} \succ \boldsymbol{x_h}$*).*

As remarked by a reviewer, this lemma leads directly to a simple recursive $O(\sigma n)$-time algorithm to compare two strings $\boldsymbol{v}$ and $\boldsymbol{x}$, where $\sigma = |\Sigma|$ and $n = |\boldsymbol{v}| + |\boldsymbol{x}|$:

1. Scan $\boldsymbol{v}$ and $\boldsymbol{x}$ separately to determine their largest letters $g_{\boldsymbol{v}}$ and $g_{\boldsymbol{x}}$, respectively, as well as corresponding frequencies $j = \nu\boldsymbol{v}$ and $k = \nu\boldsymbol{x}$.
2. If $g_{\boldsymbol{v}} \ne g_{\boldsymbol{x}}$ or $j \ne k$, then return "$\prec$" or "$\succ$" according to Lemma 3.4(1) & (2), and exit.
3. Scan $\boldsymbol{v}$ and $\boldsymbol{x}$ together to determine the least h such that $\boldsymbol{v_h} \ne \boldsymbol{x_h}$.
4. Set $\boldsymbol{v} \leftarrow \boldsymbol{v_h}$, $\boldsymbol{x} \leftarrow \boldsymbol{x_h}$ and goto (1).

The worst case $O(n^2)$ for this algorithm would be achieved by strings such as $\boldsymbol{v} = n\,n-1 \cdots 3\,2$ and $\boldsymbol{x} = n\,n-1 \cdots 3\,1$ with $\sigma = n$.

The next lemma is an extension of Lemma 3.4(3). It provides the interesting insight that if $\boldsymbol{v}$ is any proper subsequence of $\boldsymbol{x}$, then $\boldsymbol{v} \prec \boldsymbol{x}$, which of course is not the case for lexorder. (This is a property shared by the usual ordering of the integers, written in the usual way without leading zeros: any proper subsequence of an integer i, taken as an integer, is less than i.)

Lemma 3.5. *Given a string* $\boldsymbol{x}$ *of length* n, *let* $\boldsymbol{v} = \boldsymbol{x}[i_1]\boldsymbol{x}[i_2] \cdots \boldsymbol{x}[i_r]$ *for* $1 \le i_1 < i_2 < \cdots < i_r$, $0 \le r < n$. *Then* $\boldsymbol{v} \prec \boldsymbol{x}$.

Proof. Write $V_k(\boldsymbol{x})$ as in V-form equation (2), and observe that if $\boldsymbol{v}$ includes j occurrences of g, $0 \le j < k$, then $\boldsymbol{v}$ must have ancestor g^j in the star tree but no ancestor g^k. Thus in this case, from Lemma 3.4(2), $\boldsymbol{v} \prec \boldsymbol{x}$.

Suppose then that $j = k$ and write $V_k(\boldsymbol{v}) = \boldsymbol{v_0} g \boldsymbol{v_1} g \cdots \boldsymbol{v_{k-1}} g \boldsymbol{v_k}$, also in V-form (2). Since $|\boldsymbol{v}| < |\boldsymbol{x}|$, it follows that there is some least integer $k' \in 0..k$ such that $|\boldsymbol{v_{k'}}| < |\boldsymbol{x_{k'}}|$, hence that $\boldsymbol{v}$ has ancestor

$$A_{s_{k'}}(\boldsymbol{v}) = \boldsymbol{x_0^T} g \boldsymbol{x_1^T} g \cdots \boldsymbol{x_{k-1}^T} g \boldsymbol{x_k^T}, \tag{3}$$

where $s_{k'} = |\boldsymbol{v_{k'}}| + |\boldsymbol{v_{k'+1}}| + \cdots + |\boldsymbol{v_k}|$, and

$$\begin{aligned} \boldsymbol{x_h^T} &= \boldsymbol{x_h} \quad \text{for } 0 \le h < k', \\ &= \boldsymbol{\varepsilon} \quad \text{for } k' \le h \le k. \end{aligned}$$

Then $A_{s_{k'}}(\boldsymbol{v}) = \boldsymbol{z} g^{k-k'}$ for some string $\boldsymbol{z}$ that either is empty or has suffix g. On the other hand, we know from Lemma 3.3 that $\boldsymbol{x}$ has ancestor

$$A_{t_{k'}}(\boldsymbol{x}) = \boldsymbol{x_0^T} g \boldsymbol{x_1^T} g \cdots \boldsymbol{x_{k-1}^T} g \boldsymbol{x_k^T}, \tag{4}$$

where $t_{k'} = |\boldsymbol{x_{k'}}| + |\boldsymbol{x_{k'+1}}| + \cdots + |\boldsymbol{x_k}| > s_{k'}$. Comparing (3) and (4), we see that $A_{s_{k'}}(\boldsymbol{v}) = A_{t_{k'}}(\boldsymbol{x})$ is a common ancestor of $\boldsymbol{v}$ and $\boldsymbol{x}$, whereas $\overline{\boldsymbol{v}} = A_{s_{k'}-|\boldsymbol{v_{k'}}|} = \boldsymbol{z}\boldsymbol{v_{k'}} g^{k-k'}$ and $\overline{\boldsymbol{x}} = A_{t_{k'}-|\boldsymbol{x_{k'}}|} = \boldsymbol{z}\boldsymbol{x_{k'}} g^{k-k'}$ are distinct. It follows from Lemma 3.4(3) that, $\boldsymbol{v} \prec \boldsymbol{x}$ (with $|\boldsymbol{v}| < |\boldsymbol{x}|$ and largest letter g) if and only if $\boldsymbol{v_{k'}} \prec \boldsymbol{x_{k'}}$ (with $|\boldsymbol{v_{k'}}| < |\boldsymbol{x_{k'}}|$ and largest letter $g_1 < g$), and so to prove the lemma we need only consider the proper substrings $\boldsymbol{v_{k'}}$ and $\boldsymbol{x_{k'}}$. We can therefore continue recursively, at each stage replacing $\boldsymbol{v}/\boldsymbol{x}$-strings by proper $\boldsymbol{v}/\boldsymbol{x}$-substrings in which every letter is less than the largest letter occurring in the preceding stage. Since $\boldsymbol{x}$ is finite, and since the $\boldsymbol{v}$-substring is always shorter than the $\boldsymbol{x}$-substring, it must at some stage be true that the occurrences of the current largest letter in the $\boldsymbol{v}$-substring are fewer than its occurrences in the $\boldsymbol{x}$-substring. Hence $\boldsymbol{v} \prec \boldsymbol{x}$, as required. □

Although for strings $\boldsymbol{u} \prec \boldsymbol{v}$ and $\boldsymbol{w}$, it does not follow that $\boldsymbol{uw} \prec \boldsymbol{v}$, we can however extend to the right:

Lemma 3.6. *For any strings $\boldsymbol{u} \prec \boldsymbol{v}$ and $\boldsymbol{w}$, $\boldsymbol{u} \prec \boldsymbol{vw}$.*

Proof. If $\boldsymbol{w} = \varepsilon$, the result holds trivially; otherwise, applying Lemma 3.5, we have $\boldsymbol{u} \prec \boldsymbol{v} \prec \boldsymbol{vw}$. □

We now introduce a natural analogue of Lyndon words, derived from V-order, which we call *V-words*.

Definition 3.7 (*[9]*)**.** A string $\boldsymbol{w}$ over Σ is a ***V-word*** if it is the unique minimum in V-order $\prec$ among the set of rotations of $\boldsymbol{w}$.

Observe that the word *cabb* in Example 3.2 is a V-word; note also that a V-word cannot be a repetition. We give examples of V-words on the alphabet $1 < 2 < 3 < 4$ that are not Lyndon words on the same ordering:

Example 3.8. (1) The V-ordering of all rotations of the string 13142 gives $42131 \prec 14213 \prec 31421 \prec 13142 \prec 21314$. Hence 42131 is a V-word, while 13142 is a Lyndon word.
(2) 3211312 and 44124232 are both V-words, while the corresponding Lyndon words are the rotations 1131232 and 12423244, respectively.

In order to describe the structure of V-words, we introduce the idea of ***lexicographic extension*** $LEX(F)$ based on a specified factorization F of strings.

Definition 3.9 (*Modified from [9]*)**.** Suppose that according to some factorization F, two strings $\boldsymbol{u}, \boldsymbol{v} \in \Sigma^+$ are expressed in terms of nonempty factors:

$$\boldsymbol{u} = \boldsymbol{u_1 u_2} \cdots \boldsymbol{u_m}, \quad \boldsymbol{v} = \boldsymbol{v_1 v_2} \cdots \boldsymbol{v_n}.$$

Then $\boldsymbol{u} \prec_{LEX(F)} \boldsymbol{v}$ if and only if one of the following holds:

(1) $\boldsymbol{u}$ is a proper prefix of $\boldsymbol{v}$ (that is, $\boldsymbol{u_i} = \boldsymbol{v_i}$ for $1 \le i \le m < n$); or
(2) for some $i \in 1..\min(m, n)$, $\boldsymbol{u_j} = \boldsymbol{v_j}$ for $j = 1, 2, \ldots, i-1$, and $\boldsymbol{u_i} \prec \boldsymbol{v_i}$.

Of course the choice of the order that is extended from lexorder can be varied: in Definition 3.9 we have used V-order $\prec$, but for this hybrid method, lexorder with co-lexorder or any other total order could be used instead. Also the definition depends absolutely on the choice of the factorization F; for example, if $\boldsymbol{u} = \boldsymbol{v} = ab$, but we perversely choose F such that $\boldsymbol{u_1} = a, \boldsymbol{u_2} = b, \boldsymbol{v_1} = ab$, then $\boldsymbol{u} \prec_{LEX(F)} \boldsymbol{v}$! In order to make the definition useful, we choose the factors associated with every $\boldsymbol{x}$ to be $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_k}$ determined by writing $\boldsymbol{x}$ in its unique V-form (2). The pairwise comparisons of Definition 3.9 are then restricted to rotations $\boldsymbol{x_i x_{i+1}} \cdots \boldsymbol{x_k x_1} \cdots \boldsymbol{x_{i-1}}$, $1 \le i \le k$. In this case, we write *LEX* instead of $LEX(F)$. Consequently, using the fact that $\boldsymbol{x_i} \preceq \boldsymbol{x_j}$ implies $g\boldsymbol{x_i} \preceq g\boldsymbol{x_j}$, we can establish

Theorem 3.10 (*[9]*)**.** *Let $\boldsymbol{x} \in \Sigma^+$. Then $\boldsymbol{x}$ is a V-word if and only if its V-form (2) has $\boldsymbol{x_0} = \varepsilon$ with $\boldsymbol{x_1 x_2} \cdots \boldsymbol{x_k}$ a Lyndon word under lexicographic extension $\prec_{LEX}$.*

An application of Theorem 3.10 and Lemma 2.3 yields the following result:

Theorem 3.11 (*[9]*)**.** *The set of V-words forms a circ-UMFF $\mathcal{V}$.*

From now on we will denote the V-word circ-UMFF by $\mathcal{V}$.

Notice that the V-form (2) of a V-word $\boldsymbol{x}$ must, by the properties of V-order, begin with the largest letter g (over Σ) in $\boldsymbol{x}$. Thus in (2) $\boldsymbol{x_0} = \boldsymbol{\varepsilon}$, and a V-word must take the form

$$g\boldsymbol{x_1} \cdots g\boldsymbol{x_{k-1}} g\boldsymbol{x_k}. \tag{5}$$

Accordingly the k rotations of (5) (those commencing $g\boldsymbol{x_1}, g\boldsymbol{x_2}, \ldots, g\boldsymbol{x_k}$) are the only ones that need to be considered in order to determine the unique rotation of primitive $\boldsymbol{x}$ which is its V-word. Hence to test if a string $\boldsymbol{w}$ written as in (5) is a V-word it suffices to check that $\boldsymbol{x_1 x_2} \cdots \boldsymbol{x_k}$ is Lyndon under lex-extension. Further, if $\boldsymbol{w_1 w_2} \cdots \boldsymbol{w_m}$ is any maximal factorization by V-words, then by Theorem 2.9, $\boldsymbol{w_1} \succeq_{\mathcal{V}} \boldsymbol{w_2} \succeq_{\mathcal{V}} \cdots \succeq_{\mathcal{V}} \boldsymbol{w_m}$ while $\mu\boldsymbol{w_1} \le \mu\boldsymbol{w_2} \le \cdots \le \mu\boldsymbol{w_m}$. These observations suggest that Duval's classic Lyndon factorization algorithm [16] can be modified into a V-word factorization algorithm in a fairly straightforward way, as we show later.

We can extend Definition 3.9 in an obvious way to any total order $\mathcal{T}$, such that $\boldsymbol{u_i} \ll_{\mathcal{T}} \boldsymbol{v_i}$ means $\boldsymbol{u_i}$ comes before $\boldsymbol{v_i}$ in $\mathcal{T}$, which in turn gives rise to a natural generalization of Lyndon words:

Definition 3.12. Let $\mathcal{T}$ be a total order of Σ^* with the order relation $\ll_{\mathcal{T}}$. Then a string $\boldsymbol{x}$ in V-form with $\boldsymbol{x_0} = \varepsilon$ is a ***Hybrid Lyndon*** if and only if it is Lyndon under lexicographic extension $\ll_{\mathcal{T}}$.

Clearly, Hybrid Lyndons satisfy the **xyz** Lemma 2.3, but applied now to prescribed substrings $\boldsymbol{u_i}, \boldsymbol{v_j}$. It follows that V-words are an instance of Hybrid Lyndons. Further, any Hybrid Lyndon of type Flight Deck can be factored using a modification of Duval's algorithm, subject to optimizations based on particular features of the given order $\mathcal{T}$. Observe further that regular Lyndon words are also Hybrid Lyndons since, as in Theorem 3.10, we can ignore all instances of the first letter occurring in the string and compare the remaining substrings in lexorder. Moreover, we can lift the requirement that $\boldsymbol{x_0} = \varepsilon$ and choose another letter (by property like smallest/largest, or by position) in the string as a marker to distinguish the $\boldsymbol{x_i}$.

With reference to Definition 2.12, clearly the set of Lyndon words is type Flight Deck, and we now show that the set of V-words is also type Flight Deck but on an inverted alphabet.

Lemma 3.13. *The circ-UMFF $\mathcal{V}$ is type Flight Deck.*

Proof. Let $\boldsymbol{w} \in \mathcal{V}$, so that $\boldsymbol{w}$ is in V-form, with $\boldsymbol{w}[1] = g$. Then g is maximal in $\boldsymbol{w}$ with respect to V-order (that is, in Σ), but applying Lemma 3.4(1), g is minimal in $\boldsymbol{w}$ with respect to $\mathcal{V}$-order. □

Further, notice that the Flight Deck condition includes trivial cases where $\boldsymbol{w}[1 \ldots n] \in \mathcal{W}$ and, for every $1 < i \leq n$, $\boldsymbol{w}[1] <_W \boldsymbol{w}[i]$; for instance, the Lyndon word *aedcb* and the V-word *eabcd*. These kinds of simple V-words will be called *V-letters*, which we assume to be of maximum length (for instance *eabcd* rather than the prefix *eab*). Note that similarly to the Lyndon method, a factorization algorithm can just sweep across V-letters looking for an instance of $\boldsymbol{w}[1]$.

We will now show that when a V-word is in the form of a V-letter, then the set of its rotations must be in V-order (see Examples 3.2 and 3.8). But note that this property does not hold for the Lyndon circ-UMFF, as shown by the words *abc*, *acb*; nor is it the case for general V-words, as illustrated by 3132.

Lemma 3.14. *Let $\boldsymbol{w}$ be a V-letter of length n. If $\boldsymbol{r_i} = R_i(\boldsymbol{w})$, $i = 1, 2, \ldots, n$, then $\boldsymbol{r_1} \succ \boldsymbol{r_2} \succ \cdots \succ \boldsymbol{r_n} = \boldsymbol{w}$.*

Proof. Consider the rotations $\boldsymbol{r_i}$ and $\boldsymbol{r_{i+1}}$. Since there is one maximal letter g in $\boldsymbol{w}$, writing these in V-form yields $\boldsymbol{r_i} = \boldsymbol{x_{i,0}} g \boldsymbol{x_{i,1}}$ and $\boldsymbol{r_{i+1}} = \boldsymbol{x_{i+1,0}} g \boldsymbol{x_{i+1,1}}$. Then Lemma 3.4(3) applies, and so we just need to compare $\boldsymbol{x_{i,0}}$ and $\boldsymbol{x_{i+1,0}}$ in V-order. By the rotation operation, $\boldsymbol{x_{i+1,0}}$ is a subsequence (suffix) of $\boldsymbol{x_{i,0}}$, and hence Lemma 3.5 establishes that $\boldsymbol{r_i} \succ \boldsymbol{r_{i+1}}$. □

The concept of *shuffling* Lyndon words – that is, interleaving their letters – was introduced in [19,5], where it was shown that the ordered shuffle of two Lyndon words of the same length is a Lyndon word. Subsequently this operation was considered for V-words in [9], but not completed there, a problem to which we now return. We clarify that, whereas Lyndon shuffling interleaves letters, for V-words the *V-shuffle* is the process of interleaving V-letters. So we decompose a V-word which is in V-form into its V-letters; that is, strings of the form $g\boldsymbol{x_i}$.

Lemma 3.15. *Let $\boldsymbol{v} = g\boldsymbol{v_1} g \cdots \boldsymbol{v_{k-1}} g \boldsymbol{v_k}$ and $\boldsymbol{w} = g\boldsymbol{w_1} g \cdots \boldsymbol{w_{k-1}} g \boldsymbol{w_k}$ be V-words with $\boldsymbol{v} <_{\mathcal{V}} \boldsymbol{w}$. Then the ordered V-shuffle $\boldsymbol{x} = g\boldsymbol{v_1} g \boldsymbol{w_1} g \boldsymbol{v_2} g \boldsymbol{w_2} \cdots g \boldsymbol{v_k} g \boldsymbol{w_k}$ is a V-word.*

Proof. This follows from the Lyndon result, along with Definition 3.9 and Theorem 3.10. □

The Lyndon words $\boldsymbol{l_1} = aab$ and $\boldsymbol{l_2} = b$ with $\boldsymbol{l_1} <_{\mathcal{L}} \boldsymbol{l_2}$ show the necessity for words to be the same length in the shuffle. We can translate this example into ordered V-words to also show this requirement for V-words: $\boldsymbol{v} = cacacb$ and $\boldsymbol{w} = cb$. Similarly, consider the example from [9],

$$\boldsymbol{v} = 110110000, \qquad \boldsymbol{w} = 100010000, \qquad \boldsymbol{v} <_{\mathcal{V}} \boldsymbol{w},$$

whose V-shuffle is not a V-word. Writing

$$\boldsymbol{v} = 1\boldsymbol{\varepsilon}\ 10\ 1\boldsymbol{\varepsilon}\ 10000, \qquad \boldsymbol{w} = 1000\ 10000$$

in terms of their V-letters, we can see that the number of V-letters is not the same, and so Lemma 3.15 does not apply to their V-shuffle.

With reference to Definition 2.8, we now show that the order of $\mathcal{V}$ is in some cases the same as V-order while in other cases it is reversed. In particular, the alphabets are in opposite order.

Lemma 3.16. *Suppose V-order and $\mathcal{V}$ are defined over an alphabet Σ, with distinct $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{V}$.*
(1) If $\mu\boldsymbol{v} < \mu\boldsymbol{w}$, then $\boldsymbol{v} \prec \boldsymbol{w}$ and $\boldsymbol{v} >_{\mathcal{V}} \boldsymbol{w}$.
(2) If $\mu\boldsymbol{v} = \mu\boldsymbol{w}$ and $\nu\boldsymbol{v} < \nu\boldsymbol{w}$ then $\boldsymbol{v} \prec \boldsymbol{w}$ and $\boldsymbol{v} <_{\mathcal{V}} \boldsymbol{w}$ if $\boldsymbol{vw}$ is a Hybrid Lyndon.
(3) If $\mu\boldsymbol{v} = \mu\boldsymbol{w}$ and $\nu\boldsymbol{v} = \nu\boldsymbol{w}$, and if $\boldsymbol{v} \prec \boldsymbol{w}$ then $\boldsymbol{v} <_{\mathcal{V}} \boldsymbol{w}$.

Proof. (1) The V-order $\boldsymbol{v} \prec \boldsymbol{w}$ comes from Lemma 3.4(1). From Theorem 2.7(2–3), $\boldsymbol{vw}$ is primitive and either $\boldsymbol{v} <_{\mathcal{V}} \boldsymbol{w}$ or $\boldsymbol{w} <_{\mathcal{V}} \boldsymbol{v}$. Consider all rotations of $\boldsymbol{vw}$, and the associated V-form of every rotation, $\boldsymbol{x_0} g \boldsymbol{x_1} g \cdots \boldsymbol{x_{k-1}} g \boldsymbol{x_k}$. Suppose rotation $R_r(\boldsymbol{vw}) = \boldsymbol{r}$ starts with $\mu\boldsymbol{w}$ and rotation $R_{r'}(\boldsymbol{vw}) = \boldsymbol{r'}$ starts with a smaller letter. Then in their respective V-forms, $\boldsymbol{r} = \boldsymbol{\varepsilon} g \boldsymbol{x_1} g \cdots \boldsymbol{x_{k-1}} g \boldsymbol{x_k}$, $\boldsymbol{r'} = \boldsymbol{x_0'} g \boldsymbol{x_1'} g \cdots \boldsymbol{x_{k-1}'} g \boldsymbol{x_k'}$ with $\boldsymbol{x_0'} \neq \boldsymbol{\varepsilon}$, and so from Lemma 3.4(3) we have $\boldsymbol{r} \prec \boldsymbol{r'}$. So the minimum rotation of $\boldsymbol{vw}$ must start with $\mu\boldsymbol{w}$, and similarly $\boldsymbol{w}$ starts with $\mu\boldsymbol{w}$. Hence $\boldsymbol{wv} \in \mathcal{V}$, and Definition 2.6 applies. When $\boldsymbol{v}$, $\boldsymbol{w}$ are single letters, we see that these two orders on Σ are opposite.
(2) The V-order $\boldsymbol{v} \prec \boldsymbol{w}$ is Lemma 3.4(2). The order on $\mathcal{V}$ is determined by lex-extension, Definition 3.9.
(3) This is Lemma 3.4(3) together with lex-extension, Definition 3.9. □

The pairs ($\boldsymbol{v} = 4142$, $\boldsymbol{w} = 43$) and ($\boldsymbol{v} = 4243$, $\boldsymbol{w} = 41$) illustrate the weak statement of Lemma 3.16 above.

The following example demonstrates the difference between factoring a string into Lyndon words and into V-words.

Example 3.17. Let $\boldsymbol{w} = 33132421$. Then the Lyndon factorization of $\boldsymbol{w}$ is $3 \geq_{\mathcal{L}} 3 \geq_{\mathcal{L}} 13242 \geq_{\mathcal{L}} 1$, while the factorization of $\boldsymbol{w}$ into V-words is $33132 \geq_{\mathcal{V}} 421$.

Clearly, if $\lambda \in \Sigma$ then the repetition $\boldsymbol{\lambda^k}$ factors into k factors $\lambda \geq_{\mathcal{W}} \lambda \geq_{\mathcal{W}} \cdots \geq_{\mathcal{W}} \lambda$ in any $\mathcal{W}$-order. By contrast in the more general case, we establish distinctness of factorizations considered here. So for a string $\boldsymbol{w}$, let $L = L(\boldsymbol{w})$ be the Lyndon factorization of $\boldsymbol{w}$ and let $V = V(\boldsymbol{w})$ be the V-word factorization of $\boldsymbol{w}$.

Lemma 3.18. *If a string $\boldsymbol{w}$ on an ordered alphabet Σ contains at least two distinct letters, then $L(\boldsymbol{w}) \neq V(\boldsymbol{w})$.*

Proof. By hypothesis, $\boldsymbol{w}$ has prefix $\boldsymbol{w}[1..k] = \lambda_1^{k-1}\lambda_2$ for some integer $k > 1$ and letters λ_1 and $\lambda_2 \neq \lambda_1$ in Σ. If $\lambda_1 < \lambda_2$ in Σ, then $\lambda_1^{k-1}\lambda_2$ is a prefix of the first factor of $\boldsymbol{w}$ in L, while the first factor in V is λ_1. If on the other hand $\lambda_1 > \lambda_2$ in Σ, then λ_1 is the first factor of $\boldsymbol{w}$ in L, while the prefix of the first factor in V is $\lambda_1^{k-1}\lambda_2$. □

An immediate consequence of this lemma is that for any pair of words $(\boldsymbol{\ell}, \boldsymbol{v})$, $\boldsymbol{\ell} \in \mathcal{L}$, $\boldsymbol{v} \in \mathcal{V}$, and not both single letters, it follows that $\boldsymbol{\ell} \neq \boldsymbol{v}$. For the particular case of a string that decomposes into a repetition of Lyndon words, the distinct partition into V-words consists (almost) of a cyclic shift of those Lyndon words:

Lemma 3.19. *Let $\boldsymbol{w}$ be a string over Σ^+ containing at least two distinct letters. Suppose that $L(\boldsymbol{w}) = \boldsymbol{l^j}$, where $\boldsymbol{l} \in \mathcal{L}$ and $j > 1$. Then $V(\boldsymbol{w}) = V(\boldsymbol{l_p})\boldsymbol{v^k}V(\boldsymbol{l_s})$, where $\boldsymbol{v} \in \mathcal{V}$, and $V(\boldsymbol{l_p})$, $V(\boldsymbol{l_s})$ are the V-word factorizations of a proper prefix, suffix of $\boldsymbol{l}$ respectively, and $k \geq 1$.*

Proof. Let $g = \mu\boldsymbol{l}$, then since L is a repetition (not equal to $\boldsymbol{g^j}$), $\boldsymbol{l}$ starts with some non-empty prefix $\boldsymbol{l'}$ where $\mu\boldsymbol{l'} < g$. We now write $\boldsymbol{w} = \boldsymbol{l_1 l_2 \cdots l_j}$ as a sequence of V-letters, namely $\boldsymbol{l'g_1g_2 \cdots g_t}$, where each $\boldsymbol{g_i}$ is a V-letter starting with g. Let $\boldsymbol{x} = \boldsymbol{g_1g_2 \cdots g_h}$ be the substring of V-letters such that $\boldsymbol{g_h}$ is in $\boldsymbol{l_1}$, and $\boldsymbol{g_{h+1}}$ is in $\boldsymbol{l_2}$. Then the non-empty suffix of $\boldsymbol{g_h}$, $\boldsymbol{l'}$, is the prefix of $\boldsymbol{l_2}$. Since the Lyndon word $\boldsymbol{l}$ is primitive, $\boldsymbol{x}$ is primitive. So let π be the rotation of $\{1, 2, \ldots, h\}$ such that $\boldsymbol{x^r} = \boldsymbol{g_{\pi_1}g_{\pi_2} \cdots g_{\pi_h}}$ is a maximal substring of $\boldsymbol{w}$ which satisfies lex-extension. Then $\boldsymbol{w} = \boldsymbol{x_s^r}(\boldsymbol{g_{\pi_1}g_{\pi_2} \cdots g_{\pi_h}})^k\boldsymbol{x_p^r} = \boldsymbol{x_s^r v^k x_p^r}$, where $\boldsymbol{x_s^r}$, $\boldsymbol{x_p^r}$ are the suffix, prefix of $\boldsymbol{x^r}$ respectively, $\boldsymbol{x_p^r x_s^r} = \boldsymbol{x^r}$, and $\boldsymbol{x_s^r x_p^r} = \boldsymbol{l}$. From Lemma 3.18, $\boldsymbol{l} \neq \boldsymbol{v}$, so we then factor the non-empty $\boldsymbol{x_s^r} = \boldsymbol{l_p}$, $\boldsymbol{x_p^r} = \boldsymbol{l_s}$ in the obvious way, giving $V(\boldsymbol{l_p})$, $V(\boldsymbol{l_s})$. □

Using the above lemma, we see that the Lyndon factorization $(113232)^3$ has the following V-word decomposition: $(1)(1)(323211)^2(32)(32)$. Naturally we get a reciprocal result if we start with a repetition of a V-word, and deduce a repetition of a Lyndon word with a similar factorization of a prefix and suffix of the V-word.

Alternatively, by considering the string $\boldsymbol{w}$ to be a *necklace*, that is a circular string, if $L(\boldsymbol{w}) = \boldsymbol{l^j}$, then trivially by Lemma 3.18 we have the distinct $V(\boldsymbol{w}) = \boldsymbol{v^k}$.

One of the attractive features of lexorder is that if we have $\boldsymbol{v} < \boldsymbol{w}$, then we can always add a common prefix which does not alter the ordering, such as $\boldsymbol{uv} < \boldsymbol{uw}$. Also, for Lyndon words, if $\boldsymbol{v} <_{\mathcal{L}} \boldsymbol{w}$, both with minimum letter not less than λ, then $\lambda\boldsymbol{v} <_{\mathcal{L}} \lambda\boldsymbol{w}$. We get a similar result for V-words, where we add a V-letter prefix:

Lemma 3.20. *For any strings $\boldsymbol{u}$ and $\boldsymbol{v} \prec \boldsymbol{w}$ defined on Σ, all with maximum letter less than g, $g\boldsymbol{uv} <_{\mathcal{V}} g\boldsymbol{uw}$.*

Proof. By Theorem 3.10 both $g\boldsymbol{uv}$ and $g\boldsymbol{uw}$ are V-words. The result is immediate from Lemma 3.3 for $\boldsymbol{u} = \boldsymbol{\varepsilon}$, and for $\boldsymbol{v} = \boldsymbol{\varepsilon}$ it follows from the corresponding result for prefixes of general circ-UMFFs (Lemma 2.10 or Lemma 3.5). It suffices therefore to show that $\boldsymbol{uv} \prec \boldsymbol{uw}$ for nonempty $\boldsymbol{u}$ and $\boldsymbol{v}$. The proof is by induction. Suppose first that $\boldsymbol{u}$ is a single letter $\lambda < g$, and consider the sequence of deletions in the star tree that yields a common ancestor $\boldsymbol{z}$ of $\lambda\boldsymbol{v}$ and $\lambda\boldsymbol{w}$. Three outcomes are possible: $\boldsymbol{z}$ has prefix $\mu > \lambda$, $\boldsymbol{z} = \boldsymbol{\varepsilon}$, and $\boldsymbol{z}$ has prefix λ. In the first of these cases, the prefix λ has been deleted on both upward paths (from $\lambda\boldsymbol{v}$ and from $\lambda\boldsymbol{w}$) in the star tree, and so therefore at some point λ was the strictly least letter (but not necessarily unique) remaining in each of the reduced strings. Thus apart from the deletion of λ, the deletions from $\lambda\boldsymbol{v}$ and $\lambda\boldsymbol{w}$ are the same as they would have been from $\boldsymbol{v}$ and $\boldsymbol{w}$, and so $\lambda\boldsymbol{v} \prec \lambda\boldsymbol{w}$. Consider the second case. Since $\boldsymbol{z} = \boldsymbol{\varepsilon}$, the previous deletions were from strings consisting of single letters, α, β say, where distinct α, β are maximal letters in $\lambda\boldsymbol{v}$, $\lambda\boldsymbol{w}$ respectively. Suppose that one of α, β is equal to λ, for otherwise the previous argument applies. If $\beta = \lambda$, then $\alpha > \lambda$ contradicting $\boldsymbol{v} \prec \boldsymbol{w}$. If $\alpha = \lambda$, then $\beta \geq \lambda$, and due to distinctness, $\alpha < \beta$ as required. In the third case, $\boldsymbol{z} = \lambda\boldsymbol{z^r}$, where $\boldsymbol{z^r}$ is the root for $\boldsymbol{v}$ and $\boldsymbol{w}$, and therefore at the deciding positions on the two paths below the root, the same letters determine $\lambda\boldsymbol{v} \prec \lambda\boldsymbol{w}$ that also determine $\boldsymbol{v} \prec \boldsymbol{w}$. Thus the lemma holds for a single letter λ.

Suppose now that the lemma holds for every prefix $\boldsymbol{u}$ of length $\ell \geq 1$, and consider a prefix $\overline{\boldsymbol{u}}$ of length $\ell + 1$. Here there are two cases to consider: either the common ancestor $\boldsymbol{z}$ of $\overline{\boldsymbol{u}}\boldsymbol{v}$ and $\overline{\boldsymbol{u}}\boldsymbol{w}$ has prefix $\overline{\boldsymbol{u}}$, or not. If so, then $\overline{\boldsymbol{u}}$ has played no role in the decision, and $\boldsymbol{v} \prec \boldsymbol{w} \Longrightarrow \overline{\boldsymbol{u}}\boldsymbol{v} \prec \overline{\boldsymbol{u}}\boldsymbol{w}$. If not, there exist ancestors $\boldsymbol{u'v'}$ and $\boldsymbol{u''w'}$ of $\overline{\boldsymbol{u}}\boldsymbol{v}$ and $\overline{\boldsymbol{u}}\boldsymbol{w}$, respectively, such that $|\boldsymbol{u'}| = |\boldsymbol{u''}| = \ell$. In fact, since the first letter deleted from $\overline{\boldsymbol{u}}$ must be the same in both cases, we see moreover that $\boldsymbol{u'} = \boldsymbol{u''}$, and we can conclude therefore from the inductive assumption that $\boldsymbol{u'v} \prec \boldsymbol{u'w}$, hence that $\boldsymbol{uv} \prec \boldsymbol{uw}$, as required. □

Naturally, known results for Lyndon words can also be considered for Hybrid Lyndon words. The following lemma shows a straightforward generalization of the statement that, if $\boldsymbol{p}$, $\boldsymbol{s}$ are a proper prefix, suffix respectively of a Lyndon word L, then $\boldsymbol{p} <_{\mathcal{L}} L <_{\mathcal{L}} \boldsymbol{s}$. Note that in the Lyndon case we only require that $\boldsymbol{p}$ is also Lyndon (*cf.* Lemma 2.10).[2] Similarly, for V-words the prefix needs to consist of V-letters, but the suffix is unrestricted.

Lemma 3.21. *Let $\boldsymbol{w} \in \mathcal{V}$ where $\boldsymbol{w} = g\boldsymbol{x_1}g\boldsymbol{x_2} \cdots g\boldsymbol{x_k}$ in V-form (5). If $\boldsymbol{p} = g\boldsymbol{x_1}g\boldsymbol{x_2} \cdots g\boldsymbol{x_j}$, $1 \leq j < k$, and $\boldsymbol{s}$ is any proper suffix of $\boldsymbol{w}$, then $\boldsymbol{p} <_{\mathcal{V}} \boldsymbol{w} <_{\mathcal{V}} \boldsymbol{s}$ and $\boldsymbol{pws} \in \mathcal{V}$.*

[2] We apologize for the typo 'prefix' instead of 'suffix' prior to Lemma 2.5 in [12].

Proof. The prefix case is immediate from Theorem 3.10, Lyndon properties and Definition 2.6. Similarly the suffix case is immediate if the suffix consists of V-letters; that is, it starts with the letter g. Otherwise, assume the suffix does not consist only of V-letters; that is, $\boldsymbol{s} = \boldsymbol{x}_j^{\boldsymbol{s}} g\boldsymbol{x}_{j+1} \cdots g\boldsymbol{x}_k$, where $1 \le j \le k$ and $\boldsymbol{x}_j^{\boldsymbol{s}}$ is a proper suffix of $\boldsymbol{x}_j$. Then $\boldsymbol{ws} = g\boldsymbol{x}_1 g\boldsymbol{x}_2 \cdots g\boldsymbol{x}_k \boldsymbol{x}_j^{\boldsymbol{s}} g\boldsymbol{x}_{j+1} \cdots g\boldsymbol{x}_k$. To show that $\boldsymbol{ws} \in \mathcal{V}$, we will consider its rotations, and use the property that if $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ are strings and $\boldsymbol{a} \prec \boldsymbol{b}$, then $\boldsymbol{a} \prec \boldsymbol{bc}$.

We need only consider rotations which start with g, so first compare $\boldsymbol{ws}$ with the row $g\boldsymbol{x}_k \boldsymbol{x}_j^{\boldsymbol{s}} g\boldsymbol{x}_{j+1} \cdots g\boldsymbol{x}_k g\boldsymbol{x}_1 g\boldsymbol{x}_2 \cdots g\boldsymbol{x}_{k-1}$. Consider the new substring $\boldsymbol{x}_k \boldsymbol{x}_j^{\boldsymbol{s}}$. By Lyndon properties and lex-extension we have $\boldsymbol{x}_1 \prec \boldsymbol{x}_k$, and from Lemma 3.5 we have $\boldsymbol{x}_k \prec \boldsymbol{x}_k \boldsymbol{x}_j^{\boldsymbol{s}}$, hence $\boldsymbol{x}_1 \prec \boldsymbol{x}_k \boldsymbol{x}_j^{\boldsymbol{s}}$ and so the row $\boldsymbol{ws}$ is smaller in V-order. Any other rotation of $\boldsymbol{ws}$ commences with a proper suffix of $\boldsymbol{w}$ consisting of V-letters, and the result follows. □

The example of the V-word $\boldsymbol{x} = 321312$, and the V-letter prefix 321 and suffix 21312, shows that $\boldsymbol{pws} = 32132131221312$ is a V-word. However, the prefixes 3213 or 32131, which are not V-letters, yield 3213321312 or 32131321312, neither of which are V-words.

We conclude this section by studying Lemma 2.11 for V-words, which yields a pseudo-partition of the circ-UMFF $\mathcal{V}$ (it is 'pseudo' as each part must contain every letter of the alphabet). From Theorem 2.7(5) we know that we can generally split any factor into two factors. In [12] we described the structure of Lyndon words which yield a unique split: a V-word counterpart is 41424143. In contrast to factors with a unique split, Lemma 2.11 involves overlapping factors in the sense of Lemma 2.3. Comparing Lemma 3.22 with Lemma 2.11 and its following commentary also shows that not all Lyndon results carry over to Hybrid Lyndons in a straightforward way.

Lemma 3.22. *Let $\boldsymbol{xy}, \boldsymbol{yz} \in \mathcal{V}$ for some nonempty $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$.*
(1) If $\boldsymbol{x}[1] = \boldsymbol{y}[1]$ then $\boldsymbol{xy} <_V \boldsymbol{xyyz} <_V \boldsymbol{xyz} <_V \boldsymbol{yz}$.
(2) If $\boldsymbol{x}[1] <_V \boldsymbol{y}[1]$ then $\boldsymbol{xy} <_V \boldsymbol{xyz} <_V \boldsymbol{xyyz} <_V \boldsymbol{yz}$.

Proof. First note that since they are the same length, neither $\boldsymbol{xyyzxyz}$ nor $\boldsymbol{xyzxyyz}$ can be in each other's paths in the star tree of V-order. From Lemma 2.11, $\boldsymbol{xyz}, \boldsymbol{xyyz} \in \mathcal{V}$, and since $\boldsymbol{y}$ is nonempty, we can apply Theorem 2.7(3), and consider when either $\boldsymbol{xyzxyyz}$ or $\boldsymbol{xyyzxyz}$ is a V-word.
(1) In view of Lemma 2.11, we need only verify that $\boldsymbol{xyyz} <_V \boldsymbol{xyz}$. We write $\boldsymbol{xyyzxyz}$ in suitable V-form ($\boldsymbol{x_0} = \boldsymbol{\varepsilon}$) where there are two cases, depending on whether or not the suffix $\boldsymbol{z}$ of $\boldsymbol{yz}$ starts with g. If $\boldsymbol{z}$ does start with g, then the V-form is given by

$$\boldsymbol{w} = g\boldsymbol{x}_1 g\boldsymbol{x}_2 \cdots g\boldsymbol{x}_i (g\boldsymbol{y}_1 g\boldsymbol{y}_2 \cdots g\boldsymbol{y}_j)^2 g\boldsymbol{z}_1 g\boldsymbol{z}_2 \cdots g\boldsymbol{z}_k g\boldsymbol{x}_1 g\boldsymbol{x}_2 \cdots g\boldsymbol{x}_i g\boldsymbol{y}_1 g\boldsymbol{y}_2 \cdots g\boldsymbol{y}_j g\boldsymbol{z}_1 g\boldsymbol{z}_2 \cdots g\boldsymbol{z}_k,$$

where $i, j, k \ge 1$. It is required to show that $\boldsymbol{xyyzxyz} \prec \boldsymbol{xyzxyyz}$, so we compare $\boldsymbol{w}$ with the rotation

$$\boldsymbol{w}^{\boldsymbol{r}} = g\boldsymbol{x}_1 g\boldsymbol{x}_2 \cdots g\boldsymbol{x}_i g\boldsymbol{y}_1 g\boldsymbol{y}_2 \cdots g\boldsymbol{y}_j g\boldsymbol{z}_1 g\boldsymbol{z}_2 \cdots g\boldsymbol{z}_k g\boldsymbol{x}_1 g\boldsymbol{x}_2 \cdots g\boldsymbol{x}_i (g\boldsymbol{y}_1 g\boldsymbol{y}_2 \cdots g\boldsymbol{y}_j)^2 g\boldsymbol{z}_1 g\boldsymbol{z}_2 \cdots g\boldsymbol{z}_k.$$

Using lex-extension ordering, since there is a common prefix $\boldsymbol{xy}$ in $\boldsymbol{w}$ and $\boldsymbol{w}^{\boldsymbol{r}}$, we compare the substrings $g\boldsymbol{y}_1 g\boldsymbol{y}_2 \cdots g\boldsymbol{y}_j$ and $g\boldsymbol{z}_1 g\boldsymbol{z}_2 \cdots g\boldsymbol{z}_k$. Since $\boldsymbol{yz} \in \mathcal{V}$, by Theorem 2.7(1) it is border-free, hence $g\boldsymbol{y}_1 g\boldsymbol{y}_2 \cdots g\boldsymbol{y}_j \prec g\boldsymbol{z}_1 g\boldsymbol{z}_2 \cdots g\boldsymbol{z}_k$. Otherwise, if $\boldsymbol{z}$ does not contain the letter g, the comparison of the rows $\boldsymbol{w}$ and $\boldsymbol{w}^{\boldsymbol{r}}$ depends on the comparison of the substrings $\boldsymbol{y}_j \prec \boldsymbol{y}_j \boldsymbol{z}$. In this case, since $\boldsymbol{z}$ is nonempty, we apply Lemma 3.5, establishing that $\boldsymbol{y}_j \prec \boldsymbol{y}_j \boldsymbol{z}$. Hence $\boldsymbol{xyyzxyz}$ satisfies the lex-extension Theorem 3.10, as required.
(2) Here we need only verify that $\boldsymbol{xyz} <_V \boldsymbol{xyyz}$. We know that $\boldsymbol{x}[1] = \mu(\boldsymbol{xyzxyyz}) = \mu(\boldsymbol{xyyzxyz})$, and in particular $\mu = \boldsymbol{x}[1] <_V \boldsymbol{y}[1]$. We write $\boldsymbol{xyzxyyz}$ in suitable V-form, that is, $\boldsymbol{w} = g\boldsymbol{x}_1 g\boldsymbol{x}_2 \cdots g\boldsymbol{x}_i \boldsymbol{yz} g\boldsymbol{x}_1 g\boldsymbol{x}_2 \cdots g\boldsymbol{x}_i \boldsymbol{yyz}$, $i \ge 1$. So we compare $\boldsymbol{w}$ with the rotation $g\boldsymbol{x}_1 g\boldsymbol{x}_2 \cdots g\boldsymbol{x}_i \boldsymbol{yyz} g\boldsymbol{x}_1 g\boldsymbol{x}_2 \cdots g\boldsymbol{x}_i \boldsymbol{yz}$. Applying Lemma 3.5 with nonempty $\boldsymbol{y}$, we see that $\boldsymbol{x}_i \boldsymbol{yz} \prec \boldsymbol{x}_i \boldsymbol{yyz}$, and similarly $\boldsymbol{xyzxyyz}$ satisfies the lex-extension Theorem 3.10 for this case also.
Recalling that the case $\boldsymbol{x}[1] >_V \boldsymbol{y}[1]$ is not possible in a Flight Deck circ-UMFF completes the proof. □

Example 3.23. Let $\Sigma = \{1 < 2 < 3 < \cdots\}$, and so $\{1 >_V 2 >_V 3 >_V \cdots\}$.
(1) Let $\boldsymbol{xy} = 4142$ and $\boldsymbol{yz} = 4243$, hence $\boldsymbol{xyz} = 414243$ and $\boldsymbol{xyyz} = 41424243$, where $\boldsymbol{xy}, \boldsymbol{yz}, \boldsymbol{xyz}, \boldsymbol{xyyz} \in \mathcal{V}$. Then $4142 <_V 41424243 <_V 414243 <_V 4243$.
(2) Let $\boldsymbol{xy} = 4142$ and $\boldsymbol{yz} = 21$, hence $\boldsymbol{xyz} = 41421$ and $\boldsymbol{xyyz} = 414221$, where $\boldsymbol{xy}, \boldsymbol{yz}, \boldsymbol{xyz}, \boldsymbol{xyyz} \in \mathcal{V}$. Then $4142 <_V 41421 <_V 414221 <_V 21$.

Notice that in Lemma 3.22(1) we have $\boldsymbol{xyyz} <_V \boldsymbol{xyz}$, while Lemma 3.22(2) yields $\boldsymbol{xyz} <_V \boldsymbol{xyyz}$, so the conditions on $\boldsymbol{x}[1], \boldsymbol{y}[1]$ imply that it is not possible to apply the **xyz** Lemma 2.3 in such a way as to generate a bordered word. Hence we have a pseudo-partition of the **xyz** type strings in $\mathcal{V}$.

4. Linear string $\prec$ comparison

In this section we describe a new algorithm for V-order $\prec$ comparison of finite strings in worst-case time linear in their lengths, thus asymptotically the same as comparison in lexorder. We suppose throughout that any string $\boldsymbol{x}$ is defined on a finite ordered alphabet Σ of size $\sigma = |\Sigma|$.

```
  — Left-extend the suffix of nondecreasing letters, then delete.
  — Position i is next right of the deleted position δ.
function delete(x, i) : x, i, δ
prev ← x[i].left
while x[i].λ ≥ x[prev].λ do
    i ← prev; prev ← x[i].left
δ ← i; i ← x[δ].right
x[i].left ← prev; x[prev].right ← i

  — Perform n2 − n1 successive deletions.
function reduce(x2, n2, n1) : x2, i2
i2 ← n2 + 1
for j ← 1 to n2 − n1 do
    (x2, i2, δ) ← delete(x2, i2)
```

Fig. 1. Reduction of $\boldsymbol{x_2}$ to n_1 letters.

In order to simplify the description of our algorithm, we suppose without loss of generality that the letters $\lambda \in \Sigma$ are drawn from the first σ natural numbers. We represent strings $\boldsymbol{x} = \boldsymbol{x}[1..n]$ using added sentinel positions $\boldsymbol{x}[0]$ and $\boldsymbol{x}[n+1]$ for processing convenience. At each position $i \in 1..n$, we define a triple $(\lambda, \mathit{left}, \mathit{right})$, where λ is the letter at position i, $\mathit{left} = i-1$, and $\mathit{right} = i+1$; for $i = 0$, $(\lambda, \mathit{left}, \mathit{right}) = (\sigma+1, -1, 1)$ and for $i = n+1$, $(\lambda, \mathit{left}, \mathit{right}) = (\sigma+1, n, n+2)$. The elements of the triple at position i are denoted $\boldsymbol{x}[i].\lambda$, $\boldsymbol{x}[i].\mathit{left}$, and $\boldsymbol{x}[i].\mathit{right}$. Thus we represent $\boldsymbol{x}$ as a linked list to facilitate the deletion of positions in $\boldsymbol{x}$ required to locate parent nodes in the star tree used to define V-order (Definition 3.1). Note that the linked list operations consist of: comparison, reading and writing to list cells, which can be implemented in constant time.

We suppose that two unequal strings $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ are given, both defined on Σ, of lengths $n_1 > 0$ and $n_2 \geq n_1$ respectively. We describe a $\Theta(n_2)$-time algorithm to solve the following problem:

(P1) Determine whether $\boldsymbol{x_1} \prec \boldsymbol{x_2}$ or $\boldsymbol{x_1} \succ \boldsymbol{x_2}$.

Recall that in order to compare two strings in V-order, it is necessary first to identify a common ancestor $\boldsymbol{v}$ (perhaps the empty string $\boldsymbol{\varepsilon}$) in the star tree, and then to compare the children of $\boldsymbol{v}$. The common ancestor is found by performing iterated deletes in the given strings according to the following rule:

> Delete the letter at the rightmost position $i \in \{1, \ldots, n\}$ of $\boldsymbol{x}$ such that $\boldsymbol{x}[i] \leq \boldsymbol{x}[i+1]$ and $\boldsymbol{x}[i] < \boldsymbol{x}[i-1]$. (Note that the sentinel positions 0 and $n+1$ ensure the correct application of this rule.)

Since no comparison can be made until $\boldsymbol{x_2}$ is reduced to the same length as $\boldsymbol{x_1}$, we begin by performing $n_2 - n_1$ deletions from $\boldsymbol{x_2}$, as shown in Fig. 1. Then we continue by performing single deletions from *both* $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$, checking at each step to determine whether or not the current deletion has made the reduced $\boldsymbol{x_1}$ equal to the reduced $\boldsymbol{x_2}$. To facilitate this checking, we introduce the idea of the ***longest matching suffix*** (LMS): the longest suffix of the reduced $\boldsymbol{x_1}$ that matches a suffix of the reduced $\boldsymbol{x_2}$. Of course LMS is a subsequence (possibly empty) of both $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$; it is useful to identify the leftmost position in LMS as LMS-ℓ, and the position immediately to the left of LMS-ℓ as LMS-$\ell\ell$. Clearly, when LMS($\boldsymbol{x_1}, \boldsymbol{x_2}$) equals the length of the reduced strings, the common ancestor of $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ in the star tree has been found ($\boldsymbol{\varepsilon}$ when LMS is empty). We make the following observations:

Remark 4.1. A position (not LMS-ℓ) of $\boldsymbol{x_1}$ (respectively, $\boldsymbol{x_2}$) is deleted within LMS($\boldsymbol{x_1}, \boldsymbol{x_2}$) if and only if the corresponding position (also not LMS-ℓ) within $\boldsymbol{x_2}$ (respectively, $\boldsymbol{x_1}$) is also deleted. In this case LMS cannot be extended to the left, but is reduced in length by one.

Remark 4.2. LMS cannot be extended to the left if *either* LMS-ℓ is deleted in either $\boldsymbol{x_1}$ or $\boldsymbol{x_2}$ *or* positions left of LMS-$\ell\ell$ are deleted in both $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$. In the latter case, LMS is unchanged; in the former it is reduced in length by one.

Remark 4.3. The LMS may possibly (but not necessarily) be extended to the left only if *either* LMS-$\ell\ell$ is deleted in both $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ *or* LMS-$\ell\ell$ is deleted in $\boldsymbol{x_1}$ (respectively, $\boldsymbol{x_2}$) and a position left of LMS-$\ell\ell$ is deleted in $\boldsymbol{x_2}$ (respectively, $\boldsymbol{x_1}$).

These are the essential facts exploited in the function *prec* (Fig. 2) that returns `TRUE` or `FALSE` depending on whether or not $\boldsymbol{x_1} \prec \boldsymbol{x_2}$.

Lemma 4.4. *Procedure* prec *executes in time and space linear in* $|\boldsymbol{x_2}|$.

Proof. We consider the efficiency of function *prec*. In terms of space, *prec* requires only three storage locations for each position in $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$, namely the triple $(\lambda, \mathit{left}, \mathit{right})$, in addition to constant space for variables and program storage; thus its space requirement is $\Theta(n_1 + n_2) = \Theta(n_2)$. To estimate time usage, first consider the function *delete*. This function may left-extend the current suffix of nondecreasing letters, but the deleted position is always at the left end of the suffix. It follows that over all the executions of *delete* (at most $n_1 + n_2$ of them), each position in $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ is visited at most twice,

```
  — Recompute the positions ℓ1, ℓ2 that define LMS-ℓ.
function LMS(x1, ℓ1, x2, ℓ2) : ℓ1, ℓ2
i1 ← x1[ℓ1].left; i2 ← x2[ℓ2].left
while x1[i1].λ = x2[i2].λ and i1 > 0 do
    i1 ← x1[i1].left; i2 ← x2[i2].left
ℓ1 ← x1[i1].right; ℓ2 ← x2[i2].right

  — Determine whether or not x1 ≺ x2.
function prec(x1, n1, x2, n2) : boolean
(x2, i2) ← reduce(x2, n2, n1)
(ℓ1, ℓ2) ← LMS(x1, n1 + 1, x2, n2 + 1)
  — Since x1 ≠ x2, x1[ℓ1].left = 0 implies that x1 lies
  — on the upward path from x2 in the star tree.
if x1[ℓ1].left = 0 then return TRUE
i1 ← n1 + 1   — i2 already set by reduce(x2, n2, n1)
repeat
    (x1, i1, δ1) ← delete(x1, i1); (x2, i2, δ2) ← delete(x2, i2)
      — Remark 4.2: possibly LMS-ℓ was deleted in at least
      — one of x1, x2; if so, shift LMS-ℓ right in both x1, x2.
    if δ1 = ℓ1 or δ2 = ℓ2 then
        ℓ1 ← x1[ℓ1].right; ℓ2 ← x2[ℓ2].right
      — Remark 4.3: possibly LMS-ℓℓ was deleted in one
      — of x1, x2, while LMS-ℓ was not deleted in either.
    elsif ℓ1 = i1 or ℓ2 = i2 then
        (ℓ1, ℓ2) ← LMS(x1, ℓ1, x2, ℓ2)
until x1[ℓ1].left = 0   — (x1[ℓ1].left = 0 ⇔ x2[ℓ2].left = 0)
  — V-order is determined by the lexorder of the last deleted letters.
if x1[δ1].λ < x2[δ2].λ then
    return TRUE
else
    return FALSE
```

Fig. 2. Match the reduced $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ to compare $\boldsymbol{x_1} : \boldsymbol{x_2}$.

with constant-time processing corresponding to each visit. These visits comprise either the scanning of letters from right to left, or, while re-scanning from left to right, a deletion. (Any deletions within LMS are achieved by the function *delete* that updates the linked list in constant time.) Similarly, the calls to function LMS (at most n_1 altogether) go from right to left without backtracking (although LMS-ℓ can be decremented), and so each position is visited at most twice, each with constant-time processing. We conclude that the time requirement for *prec* is $\Theta(n_1 + n_2)$; that is, linear in the lengths of the compared strings. □

5. Lyndon-like factorization using *V*-order

In this section we describe an asymptotically optimal algorithm for Lyndon-like factorization using V-order. Using the basic structure of Duval's Lyndon factorization algorithm [16] (while replacing letter by substring comparison), we implement efficiencies based on features specific to V-order (Lemma 3.4), while also applying the fast function for $\prec$ string comparison of Section 4.

Thus, denoting by $VF(\boldsymbol{x})$ the unique sequence of V-words of an input string $\boldsymbol{x}$ we have:

(P2) Given a nonempty string $\boldsymbol{x}$ on a finite ordered alphabet Σ, compute $VF(\boldsymbol{x})$.

In order to compute $VF(\boldsymbol{x})$, we need to identify the unique subset of V-words $\boldsymbol{w_j}$, $j = 1, 2, \ldots, J$, such that $\boldsymbol{x} = \boldsymbol{w_1 w_2} \cdots \boldsymbol{w_J}$ and $\boldsymbol{w_j} \geq_V \boldsymbol{w_{j+1}}$ for every $j \in \{1, \ldots, J-1\}$, thus satisfying the necessary factorization criteria given by Corollary 2.4 and Theorem 2.9. Observe too that when applying lex-extension (Definition 3.9) to V-word Eq. (5), for any comparison yielding $\boldsymbol{x_i} = \boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon}$ is the least string in V-order (the root of the star tree).

Fig. 3 gives pseudocode for a linear-time algorithm VF to solve (P2), based on Duval's algorithm [16] as presented in [22]. Algorithm VF essentially considers two adjacent substrings of $\boldsymbol{x}[1..n]$: a left hand one $\boldsymbol{u}$ (that may be a repetition repeated *rep* times, and whose positions are tracked by i) of length $rep \times \ell_1$ with a prefix of k_1 g's; and a right hand one $\boldsymbol{v}$ (tracked by j) of length ℓ_2 with a prefix of k_2 g's. At each step of the algorithm j is incremented by one, and then tests are performed to determine whether as a result $\boldsymbol{u}^{rep}$ and $\boldsymbol{v}$ should be concatenated into a single V-word, or whether each of *rep* occurrences of $\boldsymbol{u}$ (and perhaps also $\boldsymbol{v}$, if g has increased) should be output as the current V-word(s), or whether no decision can currently be taken. The variable h gives the rightmost position of the last V-word output (thus one less than the starting position of

```
procedure VF(x)
  — Given the input string x$, output in ascending sequence
       the rightmost positions of every V-word in VF(x)
h ← 0  — the total length of VF already output
while h < n do  — continue as long as total VF < n
    RESET
    while x[j] ≤ g do
        if x[i] = x[j] = g then
          — Extend the g-prefix of either u or v.
            if j − i = k1 then k1 ← k1 + 1; rep ← rep + 1; ℓ1 ← ℓ1 + 1
            else
                if i − h ≤ k1 then k2 ← k2 + 1
                ℓ2 ← ℓ2 + 1; i ← i + 1
        elsif x[j] = g then
          — The copy v of u truncated by a mismatched g:
              output u^rep and restart u ← v.
            output (h, rep, ℓ1); RESET; j ← j − 1
        elsif x[i] = g then
          — u contains a mismatched g: if k2 = 0, continue u
              by incrementing ℓ1; otherwise, concatenate u ← u^rep v.
            if j = k1 + 1 then ℓ1 ← k1 + 1
            else ℓ1 ← (rep × ℓ1) + 1
            rep ← 1; i ← h + 1
            if k2 > 0 then ℓ1 ← ℓ1 + ℓ2; k2, ℓ2 ← 0
        elsif x[i] = x[j] then
          — Corresponding non-g positions in u & v are equal.
            ℓ2 ← ℓ2 + 1
            if ℓ2 = ℓ1 then rep ← rep + 1; k2, ℓ2 ← 0; i ← h + 1
            else i ← i + 1
        else
          — Unequal non-g positions: compare u & extended v.
            j′ ← j; while x[j + 1] < g do j ← j + 1
              — If u ≻ v, output u^rep; otherwise, concatenate u ← u^rep v.
            if not prec(x[h + 1..h + ℓ1], ℓ1, x[j′ − ℓ2..j], j − j′ + ℓ2 + 1) then
                output (h, rep, ℓ1)
            ℓ1 ← j − h; rep ← 1; k2, ℓ2 ← 0; i ← h + 1
        j ← j + 1
    output (h, rep, ℓ1); output (h, 1, ℓ2)
```

Fig. 3. Compute the $O(n)$ partition VF($\boldsymbol{x}$).

$\boldsymbol{u}^{rep}$). VF invokes three other routines: the linear function *prec* of Section 4, an initialization procedure RESET:

$$i \leftarrow h+1;\ j \leftarrow h+2;\ g \leftarrow \boldsymbol{x}[i]$$
$$k_1, \ell_1, rep \leftarrow 1;\ k_2, \ell_2 \leftarrow 0$$

and a procedure **output** (h, rep, ℓ) that for $\ell > 0$ outputs the rightmost positions h of V-word prefixes of $\boldsymbol{u}^{rep}\boldsymbol{v}$ (and updates the global variable h):

```
if ℓ > 0 then
  for i ← 1 to rep do
    h ← h + ℓ; output h
```

For processing convenience, the input string to procedure VF is actually $\boldsymbol{x}$\$, where \$ is a sentinel letter greater than any letter of Σ (chosen to be $\sigma + 1$ in Section 4).

Lemma 5.1. *Procedure* VF($\boldsymbol{x}$) *executes correctly in time and space linear in* $|\boldsymbol{x}|$.

Proof. Analogous to Lyndon factorization, V-word factorization maintains current candidates for factors (corresponding to the current maximum letter g) in the form $\boldsymbol{u}^{rep}\boldsymbol{v}$, subject to the following rules:

- $\boldsymbol{u} = g^{k_1}\boldsymbol{u}'$, where $\boldsymbol{u}'$ contains no letter $\lambda > g$ and no substring g^k, $k \geq k_1$;
- $\boldsymbol{v} = g^{k_2}$, $0 \leq k_2 < k_1$, or $\boldsymbol{v} = g^{k_1}\boldsymbol{v}'$, where $\boldsymbol{v}'$ is a proper prefix of $\boldsymbol{u}'$;
- if it becomes true that $\boldsymbol{v}' = \boldsymbol{u}'$, set $rep \leftarrow rep + 1$, $\boldsymbol{v} \leftarrow \boldsymbol{\varepsilon}$.

Only in the case that distinct letters $\lambda_1 \prec g$ and $\lambda_2 \prec g$ are found at corresponding positions i in $\boldsymbol{u}$ and j in $\boldsymbol{v}$, respectively, does it become necessary to call *prec* to determine whether or not $\boldsymbol{u} \prec \boldsymbol{v}$ — with $\boldsymbol{v}$ first extended to the right until some letter $\lambda \succeq g$ (possibly the sentinel \$) is found. We remark that this case can occur only if $k_2 = k_1$, with the current $\ell_2 < \ell_1$.

To establish the linearity of VF's execution time, note first that all the outputs taken together require at most $O(n)$ time. Also, the restarts that occur when $\boldsymbol{x}[j] = g \neq \boldsymbol{x}[i]$ can total at most $O(n)$ time. Otherwise, all operations performed at each increment of j require only constant time except for the call to *prec*. Observe that the total number of positions tested in *prec* over all calls is at most the sum of all the terms $j - j' + \ell_1 + \ell_2 + 1$. If a call to *prec* in procedure VF yields $\boldsymbol{u} \succ \boldsymbol{v}$, then an output occurs, and $\boldsymbol{u} \leftarrow \boldsymbol{v}$; if not, however, then $\boldsymbol{u} \leftarrow \boldsymbol{uv}$ (Theorem 2.9). In the latter case, $\boldsymbol{u}$ necessarily increases in size, so that the next call to *prec*, if it occurs, requires that correspondingly more positions need to be tested. We see that repeated consecutive invocations of *prec*, without output, must yield a worst case for total time, since each call must include all the positions already tested in previous calls. Note however that the result $\boldsymbol{u} \prec \boldsymbol{v}$ can occur only if the number of g's in $\boldsymbol{v}$ is at most equal to the number of g's in $\boldsymbol{u}$, subject to the constraint noted above that $k_2 = k_1$; thus the lengths of both $\boldsymbol{u}$ and $\boldsymbol{v}$ must approximately double at each step, as in the example

$$\boldsymbol{x} = g1g2\,g1g3\;\; g1g2g1g4\;\cdots,$$

where we first find $g1 \prec g2$, then $g1g2 \prec g1g3$, $g1g2g1g3 \prec g1g2g1g4$, and so on. This example in fact must constitute a worst case for the number of positions tested in *prec*; assuming without loss of generality that $|\boldsymbol{x}| = 2^m$, we find that the total number of positions tested will be

$$2^m + 2^{m-1} + \cdots + 2 = 2(|\boldsymbol{x}| - 1). \quad \square \tag{6}$$

6. Conclusion

In this paper we have designed new algorithms for V-words, which are analogous structures to the well-known Lyndon words. Linear V-order string comparison supports a V-word factorization algorithm with linear time and space complexities. Hence we have established that important Lyndon and V-word operations have the same asymptotic complexity.

Additionally, we have presented new combinatorial results on V-order and V-words, thus identifying similarities and differences between their inherent orders and that of lexicographic order and Lyndon words. Lexicographic order is primarily a positional ordering method, whereas V-order has the potential of defining order in terms of semantics or patterns rather than position. We suggest that applications of V-order and V-words may arise in music analysis (see [1,2] for applications of circ-UMFFs in musicology).

Future research into string factorization is suggested by the novel Hybrid Lyndons and generalized lex-extension method described here, as they are adaptable to a variety of ordering techniques. We propose a suite of binary factorization algorithms to be engineered for the 32 Block-like UMFFs defined in [9], along with binary Lyndon and binary V-word factorization. En route to more general circ-UMFFs, a ternary circ-UMFF conjectured in [11] may also be suitable for algorithmics. The existence of alternative factoring methods opens avenues for optimization problems on string partitioning.

References

[1] M. Chemillier, Periodic musical sequences and Lyndon words, Soft Comput. (8–9) (2004) 611–616. ISSN 1432–7643 (Print) 1433–7479 (Online).
[2] M. Chemillier, C. Truchet, Computation of words satisfying the rhythmic oddity property (after Simha Arom's works), Inform. Process. Lett. 86 (2003) 255–261.
[3] M. Crochemore, J. Désarménien, D. Perrin, A note on the Burrows-Wheeler transformation, Theoret. Comput. Sci. 332 (1–3) (2005) 567–572.
[4] K.T. Chen, R.H. Fox, R.C. Lyndon, Free differential calculus IV—the quotient groups of the lower central series, Ann. Math. 68 (1958) 81–95.
[5] L.J. Cummings, M.E. Mayes, Shuffled Lyndon words, ARS Combin. 33 (1992) 47–56.
[6] D.E. Daykin, Ordered ranked posets, representations of integers and inequalities from extremal poset problems, in: Graphs and Order, in: I. Rival (Ed.), Proceedings of a Conference in Banff, Canada (1984), NATO Advanced Sciences Institutes Series C: Mathematical and Physical Sciences, vol. 147, Reidel, Dordrecht-Boston, 1985, pp. 395–412.
[7] D.E. Daykin, Algorithms for the Lyndon unique maximal factorization, J. Combin. Math. Combin. Comput. 77 (2011) 65–74.
[8] T.-N. Danh, D.E. Daykin, The structure of V-order for integer vectors, in: A.J.W. Hilton (Ed.), Congressus Numerantium, vol. 113, Utilitas Mat. Pub. Inc., Winnipeg, Canada, 1996, pp. 43–53.
[9] D.E. Daykin, J.W. Daykin, Lyndon-like and V-order factorizations of strings, J. Discrete Algorithms 1 (2003) 357–365.
[10] D.E. Daykin, J.W. Daykin, Properties and construction of unique maximal factorization families for strings, Internat. J. Found. Comput. Sci. 19–4 (2008) 1073–1084.
[11] D.E. Daykin, J.W. Daykin, C.S. Iliopoulos, W.F. Smyth, Generic Algorithms for Factoring Strings (submitted for publication).
[12] D.E. Daykin, J.W. Daykin, W.F. Smyth, Combinatorics of Unique Maximal Factorization Families (UMFFs), in: R. Janicki, S.J. Puglisi, M.S. Rahman (Eds.), Special String Masters Issue, Fund. Inform. 97 (3) (2009), 295–309.
[13] D.E. Daykin, J.W. Daykin, W.F. Smyth, String comparison and Lyndon-like factorization using V-order in linear time, in: R. Giancarlo, G. Manzini (Eds.), Proc. 22nd Annual Symposium on Combinatorial Pattern Matching, in: LNCS, vol. 6661, 2011, pp. 65–76.
[14] J.W. Daykin, C.S. Iliopoulos, W.F. Smyth, Parallel RAM algorithms for factorizing words, Theoret. Comput. Sci. 127 (1) (1994) 53–67.
[15] O. Delgrange, E. Rivals, STAR: an algorithm to search for tandem approximate repeats, Bioinformatics 20–16 (2004) 2812–2820.
[16] J.P. Duval, Factorizing words over an ordered alphabet, J. Algorithms 4 (1983) 363–381.
[17] J. Gil, D.A. Scott, A bijective string sorting transform, http://bijective.dogma.net/00yyy.pdf (submitted for publication).
[18] C.S. Iliopoulos, W.F. Smyth, Optimal algorithms for computing the canonical form of a circular string, Theoret. Comput. Sci. 92 (1) (1992) 87–105.
[19] M. Lothaire, Combinatorics on Words, 2nd Edition, Cambridge University Press, Cambridge, 1997.
[20] L. Perret, A chosen ciphertext attack on a public key cryptosystem based on Lyndon words, in: Proceedings of International Workshop on Coding and Cryptography, January 2005, pp. 235–244.
[21] C. Reutenauer, Free Lie algebras, in: London Math. Soc. Monographs New Ser., vol. 7, Oxford University Press, 1993, 288 pp.
[22] Bill Smyth, Computing Patterns in Strings, Pearson, 2003, 423 pp.