Math.Comput.Sci.
DOI 10.1007/s11786-016-0285-x

**Mathematics in Computer Science**

# Indeterminate String Factorizations and Degenerate Text Transformations

**Jacqueline W. Daykin · Bruce Watson**

Received: 22 April 2014 / Revised: 26 May 2014 / Accepted: 21 June 2014

**Abstract** The data explosion problem continues to escalate requiring novel and ingenious solutions. Pattern inference focusing on repetitive structures in data is a vigorous field of endeavor aimed at shrinking volumes of data by means of concise descriptions. The Burrows–Wheeler transformation computes a permutation of a string of letters over an alphabet, and is well-suited to compression-related applications due to its invertability and data clustering properties. For space efficiency the input to the transform can be preprocessed into Lyndon factors. Rather than this classic deterministic approach for letter based strings, we consider scenarios with uncertainty regarding the data: a position in an *indeterminate* or *degenerate* string is a set of letters. We first define *indeterminate Lyndon words* and establish their associated unique string factorization; then we introduce the novel *degenerate Burrows–Wheeler transformation* which may apply the indeterminate Lyndon factorization. A core computation in Burrows–Wheeler type transforms is the linear sorting of all conjugates of the input string—we achieve this in the degenerate case with lex-extension ordering. Like the original forms, indeterminate Lyndon factorization and the degenerate transform and its inverse can all be computed in linear time and space with respect to total input size of degenerate strings. Regular molecular biological strings yield a wealth of applications of big data—an important motivation for generalizing to degenerate strings is their extensive use in expressing polymorphism in DNA sequences.

**Keywords** Degenerate biological string · Degenerate Burrows-Wheeler transform · Indeterminate Lyndon word · Indeterminate suffix array · Inverse transform · Lex-extension order · Linear

Full version of an extended abstract which appeared in the Proceedings of the 2nd International Conference on Algorithms for Big Data.

Jacqueline W. Daykin (✉)
Department of Informatics, King's College London, London, UK
e-mail: jackie.daykin@kcl.ac.uk; jackie.daykin@rhul.ac.uk

Jacqueline W. Daykin
Department of Computer Science, Royal Holloway, University of London, Egham, UK

Bruce Watson
Information Science Department, Stellenbosch University, Stellenbosch, South Africa
e-mail: bwwatson@sun.ac.za

Bruce Watson
Centre for Artificial Intelligence Research, Meraka/CSIR, South Africa

Published online: 02 February 2017

**Mathematics Subject Classification** 68R15

## 1 Introduction

The global explosion of information is progressively escalating during the twenty-first century, not only in terms of volume but also the form, the complexity, of the data itself. The reality of big data is changing our experience of the world: business practice, governmental institutions, social interaction, commerce, entertainment, security, virtual reality, communications, specialized libraries and data banks, image processing, education, public health, space research, . . . .

The big data problem is creating new challenges for capturing, storing, accessing, aggregating, transmitting, searching, securing and interpreting these huge volumes of data: many Internet and supercomputing activities typically involve quantities of data in excess of terabytes; and dynamic stream data may require processing in blocks depending on the application. Multidisciplinary action has proved fruitful in many areas of big data, for instance: computer scientists collaborating with mathematicians on information security and cryptography; and bioinformatics, the interface of molecular biology and information science, researching the computation of biological strings with possibly billions of data—the impact of advances in bioinformatics extends to human health.

The fundamental form of data is a *string*, namely a sequence of symbols over an alphabet $\Sigma$. A collection of related or collated strings is often referred to as *text*. This paper focuses on strings involving uncertainty—such strings are known as *indeterminate*, or equivalently *degenerate* strings,[1] and consist of nonempty *subsets* of letters over $\Sigma$. Algorithms for indeterminate strings were introduced in [18]; suitable data structures for these strings include the prefix table or associated feasible array.

Indeterminate strings can, for instance, model Web interface data entry by specifying a set of common error letters associated with a position in a *word* (string). The agrep utility [38] has been virtually one of the few practical algorithms available for indeterminate pattern-matching.

Computations involving degenerate strings are interesting from the combinatorial point of view. The classic longest common subsequence problem has been widely applied to problems such as file comparison programs, screen redisplay applications and computing similarity of two DNA sequences; Iliopoulos et al. [20] describe finite automata based algorithms on subsequences and supersequences of degenerate strings.

In addition to the theoretical enquiry, motivation for degenerate strings arises from bioinformatics. With degenerate biological strings, nucleotide sequences are often written using the five letter alphabet $\{A, T, G, C, N\}$, where $N$ denotes an unspecified nucleotide, that is, sequencing does not allow the identity of a nucleotide at a specific position to be inferred unambiguously—a sequence with an $N$ is known as degenerate; for instance $ANTAG$ may correspond to four different interpretations: $AATAG$, $ATTAG$, $AGTAG$ and $ACTAG$. Hence in general, a sequence with $k$ unspecified nucleotides $N$ will have $4^k$ different interpretations. Such degenerate strings can express polymorphisms in DNA/RNA sequences, for instance polymorphism in binding site sequences of a family of genes; in the computation of the homology of two biological sequences it is important to take into account a common, either specific or assumed, structure [37]. Pattern matching techniques honed to degenerate DNA/RNA sequences are designed in [19].

Our interest here is to apply indeterminacy and techniques from combinatorics on words to *data clustering*—rearranging data so as to group together, or cluster, identical forms which in turn renders the data more suitable for compression, a vital activity in big data, with specific applications in bioinformatics.

In combinatorics on words, a *Lyndon word* is defined as a (generally) finite word which is strictly minimal for the lexicographic order of its conjugacy class; the set of Lyndon words permits the unique maximal factorization of any given string [8,27]. Introduced originally by Lyndon in 1954 as *standard lexicographic sequences* [29], Lyndon words have been studied extensively and are steadily finding an increasing range of applications: string combinatorics and algorithmics [15,36], constructing bases in free Lie algebras [34], succinct suffix–prefix matching of highly

[1] Terminology: *indeterminate* is common in theoretical computer science; *degenerate* is used in molecular biology.

periodic strings [32], constructing de Bruijn sequences [16], musicology [7], computing the lexicographically smallest or largest substring in a string [3], string matching [5,11], and in relation to cryptanalysis [33]. We will be applying the Lyndon factorization to a text transformation scheme for degenerate strings.

In 1994, Burrows and Wheeler [6] introduced a transformation for textual data demonstrating, not only data clustering properties, but also suitability for block sorting. The Burrows–Wheeler transform (BWT) operates by permuting the letters of a text to obtain a modified text which may be more suitable for compression—the transform is therefore used by many text compression or compression-related applications, and some self-indexing data structures [1]. Space saving techniques with the BWT can be achieved by first factoring the input text or string into Lyndon words. Compression of data is thus usually implemented via a cascade of efficient preprocessing, transformation and coding algorithms: preprocessing by the Lyndon factorization & the BWT $\rightarrow$ Move-to-Front transform $\rightarrow$ Run-Length Encoding $\rightarrow$ statistical compressor (Arithmetic or Huffman coding).

The Burrows–Wheeler transformation, possibly with the use of Lyndon words, arises in bioinformatics in the context of next-generation sequencing (NGS) techniques. In NGS, large unknown DNA sequences are fragmented into small segments (a few dozen to several hundreds of base pairs long). This process generates masses of data, typically several million "short reads". In order to reconstruct the original DNA sequence, alignment programs attempt to align or match these reads to a reference genome. Alignment programs initially applied hashing or the suffix tree/array data structures—subsequently, efficiency in memory requirement was achieved by using the BWT. So in a bioinformatics context, the input to the BWT is a reference genome, comprising in the case of the human genome of about 3 billion DNA base pairs from the alphabet $\{A, T, G, C\}$. Software implementations based on the BWT include: SOAP2 [26], BWA [25], and Bowtie [24]. Further, word-level parallelism has been applied to preprocess the genomic sequence in massive exact unique pattern matching in genomes [2].

Motivated by the degeneracy arising from genome sequencing, we introduce here a simple modification of the classic BWT transform, the *degenerate Burrows–Wheeler transform*, which is suitable for degenerate strings—for strings containing subsets of letters, we compute both the transform and its inverse in time linear in total string size. In turn, this BWT variant proposed a modification from classic Lyndon words to words containing subsets of letters, which motivated the concepts of *indeterminate conjugacy* and *indeterminate Lyndon words*.

The original BWT computation requires the fast lexicographic sorting of all conjugates (rotations) of the input text, which can be achieved using efficient suffix-sorting techniques. We indicate how to modify the linear-time suffix array construction of Ko and Aluru [22] from regular to indeterminate strings by using lex-extension ordering.

The remainder of the paper is organized as follows. In Sect. 2 we introduce the required basic definitions. In Sect. 3 we discuss the classic Burrows–Wheeler transform. The combinatorics of indeterminate Lyndon words is given in Sect. 4. In Sect. 5 we present the degenerate Burrows–Wheeler transform along with a required indeterminate suffix array technique. Finally, we conclude and suggest some future directions in Sect. 6.

## 2 Definitions and Preliminaries

Consider a finite totally ordered alphabet $\Sigma$ of size $|\Sigma| = \sigma$ which consists of a set of characters (or letters, symbols). A *string* (*word*) is a sequence of zero or more characters over $\Sigma$. The set of all non-empty strings over $\Sigma$ is denoted by $\Sigma^+$. The empty string is the null sequence of characters (hence zero length) and is denoted by $\boldsymbol{\varepsilon}$; furthermore, $\Sigma^* = \Sigma^+ \cup \boldsymbol{\varepsilon}$. Note we write strings in mathbold such as $\boldsymbol{w}, \boldsymbol{x}$.

A string $\boldsymbol{s}$ of length $|\boldsymbol{s}| = n$ is represented by $\boldsymbol{s}[1 \dots n]$, where $\boldsymbol{s}[i] \in \Sigma$ for $1 \le i \le n$. The $i$-th symbol of a string $\boldsymbol{s}$ is denoted by $\boldsymbol{s}[i]$, or simply $\boldsymbol{s}_i$. We denote by $\boldsymbol{s}[i \dots j]$ the substring of $\boldsymbol{s}$ that starts at position $i$ and ends at position $j$, equivalently $\boldsymbol{s}_i \cdots \boldsymbol{s}_j$.

A string $\boldsymbol{w}$ is a *substring* of $\boldsymbol{s}$ if $\boldsymbol{s} = \boldsymbol{uwv}$, where $\boldsymbol{u}, \boldsymbol{v} \in \Sigma^*$; specifically a string $\boldsymbol{w} = w_1 \cdots w_m$ is a substring of $\boldsymbol{s} = s_1 \cdots s_n$ if $w_1 \cdots w_m = s_i \cdots s_{i+m-1}$ for some $i$. Words $\boldsymbol{s}[1 \dots i]$ are called *prefixes* of $\boldsymbol{s}$, and words $\boldsymbol{s}[i \dots n]$ are called *suffixes* of $\boldsymbol{s}$. The prefix $\boldsymbol{u}$ (respectively suffix $\boldsymbol{v}$) is a proper prefix (suffix) of a word $\boldsymbol{s}$ if $\boldsymbol{s} \neq \boldsymbol{u}, \boldsymbol{v}$. Words that are both prefixes and suffixes of $\boldsymbol{s}$ are called borders of $\boldsymbol{s}$. By $border(\boldsymbol{s})$ we denote the length of the longest border of $\boldsymbol{s}$ that is shorter than $\boldsymbol{s}$; if $border(\boldsymbol{s}) = 0$ then $\boldsymbol{s}$ is *border-free*. A string $\boldsymbol{s}$ is said to be *primitive* if it cannot be written as $\boldsymbol{w}^k$ with $\boldsymbol{w} \in \Sigma^+$ and $k \geq 2$, i.e., it is not a power of another string.

A string $\boldsymbol{y} = \boldsymbol{y}[1 \dots n]$ is a *conjugate* (*cyclic shift or rotation*) of $\boldsymbol{x} = \boldsymbol{x}[1 \dots n]$ if $\boldsymbol{y}[1 \dots n] = \boldsymbol{x}[i \dots n]\boldsymbol{x}[1 \dots i-1]$ for some $1 \leq i \leq n$ (for $i = 1$, $\boldsymbol{y} = \boldsymbol{x}$).

An *indeterminate* string $\boldsymbol{x} = \boldsymbol{x}[1 \dots m]$ on an alphabet $\Sigma$ is a sequence of nonempty subsets of $\Sigma$; $\boldsymbol{x}$ is equivalently known as a *degenerate* string. Specifically, an indeterminate string $\boldsymbol{x}$ has the form $\boldsymbol{x} = \boldsymbol{x_1 x_2} \cdots \boldsymbol{x_m}$, where each $\boldsymbol{x_i}$ is a set of letters over $\Sigma$, and while $|\boldsymbol{x}| = m$, computationally we will be accounting for the total size of the string, that is $||\boldsymbol{x}|| = n = \sum_{i=1}^{m} |\boldsymbol{x_i}|$; if some $|\boldsymbol{x_i}| = 1$ then this is the usual case of a single letter in a string denoted as $x_i$. So a typical instance of a degenerate string may have the form $\boldsymbol{u} = \boldsymbol{u_1} u_2 u_3 \boldsymbol{u_4} u_5 \cdots \boldsymbol{u_{m-1}} u_m$; in a regular string all sets are unit size. Moreover, with degeneracy we can allow the $\boldsymbol{x_i}$ to be multisets. We also write the sets in degenerate strings in mathbold (unless they are known to be unit size) - there is no ambiguity as regular and degenerate strings are used in different contexts here.

## 3 The Burrows–Wheeler Transform

The Burrows–Wheeler text transformation scheme was invented by Michael Burrows and David Wheeler in 1994 [6], although it is based on a previously unpublished transformation discovered by Wheeler in 1983. Interest in the transform has persisted for two decades leading to an increasing range of theoretical insights, practical applications, and software implementations [1].

The basic BWT algorithm permutes an input string $T$ (text) of $n$ characters into a transform in three conceptual stages: first the $n$ rotations (cyclic rotations or conjugates) of $T$ are formed; these rotations are then sorted lexicographically giving the $n \times n$ BWT matrix $M$; finally the last (right-most) character of each of the rotations, that is the last column of the matrix $M$, is extracted into a string $L$ (last)—specifically, the $i$th character of $L$ is the last character of the $i$th sorted rotation. In addition to $L$, the algorithm computes the index $i$ of the occurrence of the original text $T$ in the sorted list of rotations. The pair $(L, i)$ is known as the *transform*, that is BWT($T$) = $(L, i)$. Furthermore, the BWT can be constructed efficiently since the heart of the computation is sorting the rotations which, by applying a fast suffix-sorting technique such as [22], can be achieved in linear time. It is the data clustering properties of this transform, usually exhibiting long runs of identical characters, together with the fact that it is invertible, that has sparked so much interest.

Given only $L$ and the index $i$, the original text $T$ can be reconstructed in linear time [6]. For this, observe that every row and every column of $M$, and hence also the transformed text $L$, is a permutation of $T$. In particular, the first column $F$ of $M$, can be obtained by lexicographically sorting the characters of $L$ (or, equally, the characters of $T$). By constructing a Hamiltonian cycle of $L$ and $F$, the Last-First Mapping, we can recover the input—commencing with $L[i]$, if $L[i]$ is the $k$th occurrence of $L[i]$ in $L$, then we map to $F[j]$ the $k$th occurrence of $L[i]$ in $F$, and then to $L[j]$; and repeat until we get back to $L[i]$. Hence, importantly, the original text can be recovered *losslessly*; therefore, following transformation, the data is primed for lossless compression methodologies.

A simple observation shows that, since by definition a Lyndon word is the strictly least amongst its conjugates, if the input text forms a Lyndon word, then the index $i$ will be 1 and therefore redundant, thus offering a space saving of $O(\log n)$ bits. Accordingly, BWT variants have been considered: Scott followed by Kufleitner introduced the bijective multi-word BWT; Schindler and later Kufleitner proposed the bijective sort transform—these variants are based on the Lyndon factorization of the input [17,23].

In practice, bzip2 is an open source file compressor that uses the BWT: bzip2 compresses data in blocks of size between 100 and 900 kilobytes by first applying the BWT followed by the Move-to-Front transform and then Huffman coding (originally Arithmetic coding).

The BWT has also been implemented in bioinformatics due to the repetitions inherent in biological sequences. In an effort to reduce the memory requirement with hashing-based sequence alignment, several alignment utilities were developed that use the BWT: SOAP2 [26], BWA [25], and Bowtie [24]. Furthermore, NGS sequencing technologies have resulted in collections of hundreds of millions of DNA sequences. Knowing the longest common prefix array

(LCP) of such a collection would facilitate the rapid computation of biologically meaningful information such as: maximal exact matches, shortest unique substrings and shortest absent words. Although CPU-efficient algorithms for computing the LCP of a string exist, they require the presence in RAM of large data structures, hence clearly infeasible for NGS datasets. In [4] the first lightweight method is proposed that simultaneously computes, via sequential scans, the LCP and BWT of very large collections of sequences. Computational results on collections as large as 800 million 100-mers demonstrated that their algorithm scales to the vast sequence collections encountered in human whole genome sequencing experiments.

Common usage with text files involves taking a file $T$ and modifying it. Previously, a major bottleneck with this scenario was the fact that the BWT($T$) had to be entirely reconstructed from scratch whenever $T$ was modified. In [35], a four-stage algorithm is presented that updates the BWT($T$) when the text $T$ is modified into $T'$ according to standard edit operations—insertion, deletion, and substitution of a character or a factor. Along with the algorithm that directly converts BWT($T$) into BWT($T^{'}$), they also sketch a method for converting the suffix array of $T$ into the suffix array of $T^{'}$. Although the worst-case time complexity is $O(|T|\log|T|(1+\log\sigma/\log\log|T|))$, experimentally their algorithm is shown to be competitive in practice.

Other notable contributions to the BWT research arena include: the BWT is shown to be a special case of the Gessel and Reutenauer transformation [9]; a generalization of the BWT suitable for a multiset of words and applied to the problem of the whole mitochondrial genome phylogeny is given in [31]; slashing the time for the BWT inversion is presented in [21]; and a constant-space comparison-based algorithm for computing the BWT is proposed in [10].

## 4 Indeterminate Lyndon Words

Lyndon words were introduced by Roger Lyndon in 1954 under the name of *standard lexicographic sequences* in order to describe a basis of free Lie algebras [28–30,34]. A *Lyndon word* is a primitive and border-free word which is strictly minimal for the lexicographical order of its conjugacy class—let $\mathcal{L}$ denote the set of Lyndon words over the finite totally ordered alphabet $\Sigma$ (although an unbounded alphabet can also be considered). These patterned words exhibit many interesting properties [27], including:

**Proposition 1** [15] *A word $\boldsymbol{w} \in \Sigma^+$ is a Lyndon word if and only if it is lexicographically less than each of its proper suffixes.*

**Proposition 2** [15] *A word $\boldsymbol{w} \in \Sigma^+$ is a Lyndon word if and only if either $\boldsymbol{w} \in \Sigma$ or $\boldsymbol{w} = \boldsymbol{uv}$ with $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{L}$, $\boldsymbol{u} < \boldsymbol{v}$.*

Importantly, the set $\mathcal{L}$ of Lyndon words permits the unique maximal factorization of any given string, unpinning the wealth of applications.

**Theorem 1** [8] *Any word $\boldsymbol{w} \in \Sigma^+$ can be written uniquely as a non-increasing product $\boldsymbol{w} = \boldsymbol{u}_1\boldsymbol{u}_2\cdots\boldsymbol{u}_k$ of Lyndon words.*

This unique decomposition of any word into Lyndon words $\boldsymbol{u}_1 \geq \boldsymbol{u}_2 \geq \cdots \geq \boldsymbol{u}_k$ facilitates divide and conquer techniques for stringology applications, for instance with variants of the BWT—see Sects. 1 and 3.

For a non-letter Lyndon word $\boldsymbol{w}$ ($\boldsymbol{w} \notin \Sigma$), the pair $(\boldsymbol{u}, \boldsymbol{v})$ of Lyndon words such that $\boldsymbol{w} = \boldsymbol{uv}$ with $\boldsymbol{v}$ of maximal length is called the *standard factorization* of $\boldsymbol{w}$. In 1983, Duval [15] developed an algorithm for Lyndon decomposition that runs in $\Theta(n)$ time and constant additional space—the algorithm cleverly iterates over a string detecting the current longest Lyndon word; when it finds one, it adds it to the results list and proceeds to search in the remaining part of the string.

A generalization of Lyndon words to circ-UMFFs has been proposed in [13]. We now introduce a particular class, the set $\mathcal{IL}$ of *indeterminate Lyndon words*—a natural extension to classic Lyndon words defined for indeterminate strings. Given an indeterminate string $\boldsymbol{x} = \boldsymbol{x_1x_2}\cdots\boldsymbol{x_m}$, the first step in defining these new Lyndon words is to assign an order to each of the sets $\boldsymbol{x_i}$ (which are not necessarily distinct). So for each $1 \leq i \leq m$, let $\underline{\boldsymbol{x_i}}$ denote the lexicographic ordering of $\boldsymbol{x_i}$ (the letters are lined up in the given alphabet order) written as a string. For example, if $\boldsymbol{x_i} = \{c, a, t, g\}$ then $\underline{\boldsymbol{x_i}} = acgt$. Hence, under the convention that the order of elements in a set doesn't matter, we have a bijective mapping $\mathcal{G} : \boldsymbol{x_i} \to \underline{\boldsymbol{x_i}}$ for $1 \leq i \leq m$, or simply $\mathcal{G} : \boldsymbol{x} \to \underline{\boldsymbol{x}}$. Furthermore, we can allow multisets under this mapping. Note that if $||\boldsymbol{x}|| = n$, and if we assume the range of letters in the alphabet is $O(n)$, an array of length $|\Sigma|$ suffices to map the given alphabet onto an integer alphabet $\{1, 2, ..., k\}$, $k \leq n$. Therefore each of the subsets $\boldsymbol{x_i}$ can be sorted in time $O(|\boldsymbol{x_i}|)$; hence the total time to compute $\underline{\boldsymbol{x}}$ is $O(n)$. Note that the extensibility of lexicographic ordering allows for more complex indeterminate structures, such as nested sets $\boldsymbol{x_i}$ of sets or substrings.

We can now state a required definition, *lex-extension order*, for the lexicographic order of given indeterminate strings $\boldsymbol{u}, \boldsymbol{v}$ over $\Sigma$.

**Definition 1** [12,14] Suppose that according to some factorization $\mathcal{F}$, two strings $\boldsymbol{u}, \boldsymbol{v} \in \Sigma^+$ are expressed in terms of nonempty factors (substrings): $\boldsymbol{u} = \boldsymbol{u}_1\boldsymbol{u}_2 \cdots \boldsymbol{u}_m$, $\boldsymbol{v} = \boldsymbol{v}_1\boldsymbol{v}_2 \cdots \boldsymbol{v}_n$. Then $\boldsymbol{u} <_{LEX(F)} \boldsymbol{v}$ 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 $1 \leq i \leq m < n$); or
(2) for some $i \in 1..min(m, n)$, $\boldsymbol{u_j} = \boldsymbol{v_j}$ for $j = 1, 2, ..., i - 1$, and $\boldsymbol{u_i} < \boldsymbol{v_i}$ (in lexicographic order).

In the case of an indeterminate string $\boldsymbol{x} = \boldsymbol{x_1x_2} \cdots \boldsymbol{x_m}$, the factorization $\mathcal{F}$ is given by the subsets $\boldsymbol{x_1x_2} \cdots \boldsymbol{x_m}$ mapped to $\underline{\boldsymbol{x_1}}\underline{\boldsymbol{x_2}} \cdots \underline{\boldsymbol{x_m}}$; for brevity we will write $\boldsymbol{u} <_{LEX} \boldsymbol{v}$. We can now proceed to clarify the concept of conjugacy for an indeterminate string.

**Definition 2** An indeterminate string $\boldsymbol{y} = \boldsymbol{y_1y_2} \cdots \boldsymbol{y_m}$ is an *indeterminate conjugate* (cyclic shift or rotation) of an indeterminate string $\boldsymbol{x} = \boldsymbol{x_1x_2} \cdots \boldsymbol{x_m}$ if $\boldsymbol{y}[1 \ldots m] = \boldsymbol{x}[i \ldots m]\boldsymbol{x}[1 \ldots i-1]$ for some $1 \leq i \leq m$ (for $i = 1$, $\boldsymbol{y} = \boldsymbol{x}$).

**Definition 3** An indeterminate string $\boldsymbol{x}$ over $\Sigma$ is an *indeterminate Lyndon word* if it is strictly minimal for the lex-extension order of its indeterminate conjugacy class under the mapping $\mathcal{G} : \boldsymbol{x} \to \underline{\boldsymbol{x}}$.

For the indeterminate Lyndon word $\boldsymbol{x} = aab\boldsymbol{uv}$ over $\Sigma = \{a < b < \cdots < z\}$, with multisets $\underline{\boldsymbol{u}} = aa$ and $\underline{\boldsymbol{v}} = aab$, to compare the indeterminate conjugates $\boldsymbol{x} = aab\underline{\boldsymbol{u}}\underline{\boldsymbol{v}}$ and $\boldsymbol{y} = \underline{\boldsymbol{u}}\underline{\boldsymbol{v}}aab$ in lex-extension order, the first comparison is $a?aa$ giving $a < \underline{\boldsymbol{u}}$ and hence $\boldsymbol{x} <_{LEX} \boldsymbol{y}$; similarly, to compare the conjugates $\boldsymbol{x} = aab\underline{\boldsymbol{u}}\underline{\boldsymbol{v}}$ and $\boldsymbol{z} = b\underline{\boldsymbol{u}}\underline{\boldsymbol{v}}aa$ in lex-extension order, the first comparison is $a?b$ giving $a < b$ and so $\boldsymbol{x} <_{LEX} \boldsymbol{z}$. Furthermore, unlike the string $aab\underline{\boldsymbol{u}}\underline{\boldsymbol{v}}$ written as a regular string with individual letters, namely $aabaaaab$ which has border $aab$, indeterminate $\boldsymbol{x}$ is border-free.

The simplicity of the definition of an indeterminate Lyndon word means that Duval's classic linear Lyndon factorization algorithm [15] extends directly to the indeterminate case: letter comparisons are replaced by those of the form $\underline{\boldsymbol{x_i}}?\underline{\boldsymbol{x_j}}$ which can each be decided in at most linear time - hence $O(n)$ overall.

Equipped with this definition we can trivially derive results analogous to those for the classic case, basically by replacing $<$ with $<_{LEX}$—we give some examples, where $\mathcal{IL}$ denotes the set of indeterminate Lyndon words.

**Proposition 3** *A word $\boldsymbol{w} \in \Sigma^+$ is an indeterminate Lyndon word if and only if it is less in lex-extension order than each of its proper suffixes.*

**Proposition 4** *A word $\boldsymbol{w} \in \Sigma^+$ is an indeterminate Lyndon word if and only if either $\boldsymbol{w}$ is a single set of elements or $\boldsymbol{w} = \boldsymbol{uv}$ with $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{IL}$, $\boldsymbol{u} <_{LEX} \boldsymbol{v}$.*

We conclude this section by establishing that these extended Lyndon words can be applied to uniquely factor indeterminate strings. A subset $\mathcal{W}$ of $\Sigma^+$ is known as a factorization family (FF) if and only if for every nonempty string $\boldsymbol{x}$ on $\Sigma$ there exists a factorization of $\boldsymbol{x}$ over $\mathcal{W}$—note that $\Sigma \subseteq \mathcal{W}$. We proceed to show that the set of indeterminate Lyndon words forms an UMFF *(unique maximal factorization family)* [12].

**Lemma 1** (The **xyz** Lemma [12]) *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 $\boldsymbol{xyz} \in \mathcal{W}$.*

**Lemma 2** *The set $\mathcal{IL}$ of indeterminate Lyndon words forms an UMFF.*

*Proof* We apply Lemma 1. Given the indeterminate strings $\boldsymbol{x}$ and $\boldsymbol{y}$, suppose $\boldsymbol{xy}, \boldsymbol{yz} \in \mathcal{IL}$ for some nonempty $\boldsymbol{y}$. According to Definition 3, we have assumed the mappings $\mathcal{G} : \boldsymbol{x} \to \underline{\boldsymbol{x}}$ and $\mathcal{G} : \boldsymbol{y} \to \underline{\boldsymbol{y}}$, so we have $\underline{\boldsymbol{xy}}, \underline{\boldsymbol{yz}} \in \underline{\mathcal{IL}} = \mathcal{IL}$ for some nonempty $\underline{\boldsymbol{y}}$, and we need to show that $\underline{\boldsymbol{xyz}} \in \underline{\mathcal{IL}}$; assume that $\underline{\boldsymbol{x}}, \underline{\boldsymbol{z}} \neq \varepsilon$, for otherwise it holds trivially. Applying lex-extension ordering $<_{LEX}$ and Proposition 3 we have $\underline{\boldsymbol{x}} <_{LEX} \underline{\boldsymbol{xy}} <_{LEX} \underline{\boldsymbol{y}} <_{LEX} \underline{\boldsymbol{yz}} <_{LEX} \underline{\boldsymbol{z}}$.

Applying Proposition 4, if either $\underline{\boldsymbol{x}}$ or $\underline{\boldsymbol{y}}$ belong to $\underline{\mathcal{IL}}$, then $\underline{\boldsymbol{xyz}} \in \underline{\mathcal{IL}}$. Otherwise, simply applying $\underline{\boldsymbol{x}} <_{LEX} \underline{\boldsymbol{y}} <_{LEX} \underline{\boldsymbol{z}}$ from above shows that the minimal conjugate is $\underline{\boldsymbol{xyz}}$.

Finally, using the bijectivity of $\mathcal{G}$ it follows that $\boldsymbol{xyz} \in \mathcal{IL}$ as required—hence the set $\mathcal{IL}$ forms an UMFF. □

**Definition 4** [13] An UMFF $\mathcal{W}$ over $\Sigma^+$ is a *circ-UMFF* if and only if it contains exactly one rotation of every primitive string $\boldsymbol{x} \in \Sigma^+$.

This definition extends naturally to indeterminate strings $\boldsymbol{x} \in \Sigma^+$. Given the indeterminate UMFF $\mathcal{IL}$, the chosen rotation for $\boldsymbol{x} \in \mathcal{IL}$ is the one that is the strict minimum in lex-extension order (applying the mapping $\mathcal{G}$ as usual); the uniqueness in Definition 3 shows that an indeterminate Lyndon word is primitive. Hence we are introducing here a new circ-UMFF, namely the set $\mathcal{IL}$ of *indeterminate Lyndon words*. Furthermore, classic properties of these Lyndon-patterned words such as ordered concatenation, border-freeness and partitioning into sub-words then follow from Theorem 3.1 in [13].

We comment that other ordering methods, that is ordered alphabets, could be applied to the mapping of the subsets $\mathcal{G} : \boldsymbol{x_i} \to \underline{\boldsymbol{x_i}}$ in order to arrange the letters as a string. This leads to a class of indeterminate Lyndon-like words $\mathcal{IL}_\omega$ for total orderings $\omega$—here we have considered $\mathcal{IL}_{Lex}$ (simply $\mathcal{IL}$) namely the set of words depending on lexicographic order of indeterminate subsets; hence, as defined, each $\underline{\boldsymbol{x_i}}$ is also a Lyndon word.

## 5 A Degenerate Burrows–Wheeler Transform

Having formalized the concept of an indeterminate Lyndon word, we are now ready to introduce a variant of the Burrows–Wheeler transform tailored to degenerate strings. The *degenerate Burrows–Wheeler transform*—denoted $D$-BWT — is a very simple extension of the original transformation, which relies only on further use of lexicographic ordering (although as mentioned at the end of Sect. 4 different orderings could be considered). Also for the analysis below, note that the total string size of $\boldsymbol{x}$ is $||\boldsymbol{x}|| = n$—see Sect. 2.

Given a degenerate string $\boldsymbol{x} = \boldsymbol{x}[1 \dots m] = \boldsymbol{x_1 x_2} \cdots \boldsymbol{x_m}$, to construct the $D$-BWT, we first perform all the mappings $\mathcal{G} : \boldsymbol{x_i} \to \underline{\boldsymbol{x_i}}$ specified in Sect. 4 in linear time. As in the original BWT transformation, we will generate the sorted rotations—the $D$-BWT matrix—of the input string, text or reference genome, depending on the application. To do this we apply a fast suffix-sorting algorithm tweaked to handle subsets rather than individual letters.

### 5.1 Indeterminate Suffix Array

The well-known suffix array of a string records the lexicographically sorted list of all of its suffixes; this data structure was originally defined for regular strings of letters whereas our interest is subsets of letters. Among the many versatile uses of the suffix array it enables the cyclic rotations of a string to be sorted into lexicographic order in linear time—this technique is typically applied in the computation of the classic transform when constructing the BWT matrix. Likewise, for the new degenerate $D$-BWT it is required to sort the rotations of an indeterminate string efficiently into lex-extension order which we will now describe.

For constructing the indeterminate suffix array we will apply a linear-time and space efficient method such as that of Ko and Aluru [22] given for the original data structure. Given an indeterminate string $\boldsymbol{x}$, we first perform the $\mathcal{G}$ mappings from $\boldsymbol{x} = \boldsymbol{x_1 x_2} \cdots \boldsymbol{x_m}$ to $\underline{\boldsymbol{x}} = \underline{\boldsymbol{x_1}}\ \underline{\boldsymbol{x_2}} \cdots \underline{\boldsymbol{x_m}}$ with $||\boldsymbol{x}|| = n$. Note that an indeterminate suffix of $\underline{\boldsymbol{x}}$ has the form $\underline{\boldsymbol{x_i}}\ \underline{\boldsymbol{x_{i+1}}} \cdots \underline{\boldsymbol{x_m}}$, and the indexes in the array will be a subset of $\{1, 2, \dots, n\}$.

Clearly, the Ko-Aluru suffix array technique can be carefully modified from $<$ to $<_{LEX}$ so as to construct an indeterminate suffix array. Alternatively, given $\underline{\boldsymbol{x}}$, we first perform a pre-sorting of the substrings $\underline{\boldsymbol{x_1}}, \underline{\boldsymbol{x_2}}, \cdots, \underline{\boldsymbol{x_m}}$

into lex-extension order, resulting in a re-labelling $\pi_1\pi_2\cdots\pi_m$ of $\underline{\boldsymbol{x}}$, where each $\pi_i$ is just a letter or ordinal number. For example, $\underline{\boldsymbol{x}} = \{abc\}\{e\}\{ad\}\{abc\}\{bce\} \rightarrow ADBAC$. This can be achieved using Bucket Sort on the finite ordered alphabet $\Sigma$, with the buckets labelled by the characters in $\Sigma$. This process is repeated in each bucket where the length of each $\underline{\boldsymbol{x_i}}$ is $O(n)$ - hence $O(n)$ overall (see details of this kind of linear sorting in [22]).

The indeterminate string $\boldsymbol{x}$ has now been re-labelled as a string of letters $\pi_1\pi_2\cdots\pi_m$ each according to their lex-extension order in $\boldsymbol{x}$. Therefore we can straightforwardly apply an existing linear letter-based suffix-sorting technique to yield a suffix array for the indexes $i \in \{1\ldots m\}$. A trivial mapping of each array element $i \rightarrow \sum_{j=1}^{i-1}|\boldsymbol{x_j}|+1$ then gives the required indeterminate suffix array. The overall linear - $O(n)$ - time and space complexities follow from the original $O(m)$ method (for instance [22]) along with $O(n)$ total string length.

### 5.2 Multi-word Degenerate Burrows–Wheeler Transform

Given the $D$-BWT matrix, as constructed from the tailored suffix array in Sect. 5.1, in the degenerate case the transform is the last right-most column of ordered subsets, specifically a permutation of $\underline{\boldsymbol{x}} = \underline{\boldsymbol{x_1}}\underline{\boldsymbol{x_2}}\cdots\underline{\boldsymbol{x_m}}$, together with the index of the row of the given input text in the matrix. However, using the re-labelling to letters $\pi_i$, the transform can be encoded as letters and hence the inverse can be achieved using the classic method - the linear Last-First mapping. Finally, the inverse mappings $\pi_i \rightarrow \boldsymbol{x_j} \rightarrow \boldsymbol{x_j}$ reconstruct the original degenerate string, hence overall linear.

For our example, $\underline{\boldsymbol{x}} = \{abc\}\{e\}\{ad\}\{a\overline{bc}\}\{bce\}$, the $D$-BWT $= ((\{a,d\}\{b,c,e\}\{e\}\{a,b,c\}\{a,b,c\}), 2)$, and so we have clustered occurrences of the set $\{a,b,c\}$ in the given indeterminate string.

Furthermore, if we assume that the input text has been factored into indeterminate Lyndon words, then this avoids indexes to the input text. Once factored, and again using the re-labelling to letters $\underline{\boldsymbol{x_i}} \rightarrow \pi_j$, the bijective multi-word BWT described by Kufleitner [23] can be applied directly—form the ordered matrix for each Lyndon factor; sort all matrices together giving the transform as the last column; the inverse transform is formed from Lyndon seeds of cyclic words—followed by inverse mappings from the $\pi_j$ to recover the subsets in the indeterminate input text.

## 6 Conclusion

In this paper we have contributed to the theory of indeterminate strings. We defined *indeterminate Lyndon words*, a simple variant of the classic Lyndon word, and showed that the set $\mathcal{IL}$ of indeterminate Lyndon words forms a new circ-UMFF—hence establishing that any indeterminate string can be uniquely factored into indeterminate Lyndon words.

We further introduced a degenerate variant of the Burrows–Wheeler transformation suitable for indeterminate strings—the *degenerate Burrows–Wheeler transform*, $D$-BWT, which may also apply the linear indeterminate Lyndon factorization to preprocess the input as a space-saving trick.

The degenerate transform required a slight adaption of the classic suffix array to achieve an *indeterminate suffix array*. However, by mapping the degenerate text from subsets to letters, and applying the linear Ko-Aluru suffix array construction [22], we achieved a linear time and space construction for the indeterminate suffix array. Hence the degenerate transform can be constructed and inverted in linear time and space.

We briefly mention $V$-words, analogous structures to Lyndon words—interestingly, they are formed using two distinct total orderings, namely lexicographic and $V$-order: strings of substrings are processed in lexicographic order while the individual substrings are compared pairwise using $V$-order (for details see [14]). Indeed, $V$-order has recently been applied to yield a new transform, the $V$-BWT [14], which also necessitated modifying the fast suffix-sorting of Ko and Aluru [22]. Similarly to indeterminate Lyndon words, indeterminate $V$-words could also be defined along with an associated degenerate $V$-BWT. However, indications are that such a complex structure (strings of sets of basic $V$-words) would be too cumbersome for practical purposes, hence we don't pursue it currently.

For future research, we note that not only does this simple extension from classic letter-based Lyndon words to those for sets open possibilities of new Lyndon-related applications involving uncertainty associated with the data, but additionally, other combinatorially interesting variations are now possible. For instance, different orderings on the alphabet could be applied to map the sets $\boldsymbol{x_i}$ to $\underline{\boldsymbol{x_i}}$; additionally, we could specify conditions so that under the mapping of sets to strings, the new string $\underline{\boldsymbol{x}}$ satisfies a property, such as being border-free, or exhibiting the $V$-word pattern, say. In particular, the mappings $\underline{\boldsymbol{x_i}}$ also allow for selection type problems, such as selecting the $i^{th}$ largest element in the subset—potentially corresponding to choosing a letter for the unspecified degenerate nucleotide $N$ in biological strings. This area, along with associated modifications to suffix arrays and transforms, is open to investigation.

## References

1. Adjeroh, D., Bell, T., Mukherjee, A.: The Burrows–Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching. Springer, NewYork (2008)
2. Antoniou, P., Daykin, J.W., Iliopoulos, C.S., Kourie, D., Mouchard, L., Pissis, S.P.: Mapping uniquely occuring short sequences derived from high throughput technologies to a reference genome. In: Proceedings of the 9th IEEE International Conference on Information Technology and Applications in Biomedicine (ITAB 2009). (2009). doi:10.1109/ITAB.2009.5394394
3. Apostolico, A., Crochemore, M.: Fast parallel Lyndon factorization with applications. Math. Syst. Theory **28**(2), 89–108 (1995)
4. Bauer, M.J., Cox, A.J., Rosone, G., Sciortino, M.: Lightweight LCP construction for next-generation sequencing datasets. CoRR. arXiv:1305.0160 (2013)
5. Breslauer, D., Grossi, R., Mignosi, F.: Simple real-time constant-space string matching. In: Giancarlo, R., Manzini, G. (eds.) CPM, volume 6661 of Lecture Notes in Computer Science, pp. 173–183 (2011)
6. Burrows, M., Wheeler, D.J.: A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation (1994)
7. Chemillier, M.: Periodic musical sequences and Lyndon words. Soft Comput. **8**(9), 611–616 (2004)
8. Chen, K.T., Fox, R.H., Lyndon, R.C.: Free differential calculus IV—the quotient groups of the lower central series. Ann. Math. **68**, 81–95 (1958)
9. Crochemore, M., Désarménien, J., Perrin, D.: A note on the Burrows–Wheeler transformation. Theor. Comput. Sci. **332**(1–3), 567–572 (2005)
10. Crochemore, M., Grossi, R., ärkkäinen, J.K., Landau, G.M.: A constant-space comparison-based algorithm for computing the Burrows–Wheeler transform. In: Proceedings of the 24th Annual Symposium on Combinatorial Pattern Matching (CPM), pp. 74–82 (2013)
11. Crochemore, M., Perrin, D.: Two-way string matching. J. ACM **38**(3), 651–675 (1991)
12. Daykin, D.E., Daykin, J.W.: Lyndon-like and V-order factorizations of strings. J. Discrete Algorithms **1**, 357–365 (2003)
13. Daykin, D.E., Daykin, J.W.: Properties and construction of unique maximal factorization families for strings. Int. J. Found. Comput. Sci. **19**(4), 1073–1084 (2008)
14. Daykin, J.W., Smyth, W.F.: A bijective variant of the Burrows–Wheeler transform using V-order. Theor. Comput. Sci. **531**, 77–89 (2014)
15. Duval, J.-P.: Factorizing words over an ordered alphabet. J. Algorithms **4**(4), 363–381 (1983)
16. Fredricksen, H., Maiorana, J.: Necklaces of beads in k colors and k-ary de Bruijn sequences. Discrete Math. **23**(3), 207–210 (1978)
17. Gil, J.Y., Scott, D.A.: A bijective string sorting transform. CoRR. arXiv:1201.3077 (2012)
18. Holub, J., Smyth, W.F.: Algorithms on indeterminate strings. In: Proceedings of the 14th Australasian Workshop on Combinatorial Algorithms (AWOCA), pp. 36–45 (2003)
19. Iliopoulos, C., Mouchard, L., Rahman, M.: A new approach to pattern matching in degenerate DNA/RNA sequences and distributed pattern matching. Math. Comput. Sci. **2**(4), 557–569 (2008)
20. Iliopoulos, C., Rahman, M., Voráček, M., Vagner, L.: Finite automata based algorithms on subsequences and supersequences of degenerate strings. J. Discrete Algorithms **8**(2), 117–130 (2010)
21. Kärkkäinen, J., Kempa, D., Puglisi, S.J.: Slashing the time for BWT inversion. In: Proceedings of the Data Compression Conference (DCC), pp. 99–108 (2012)
22. Ko, P., Aluru, S.: Space efficient linear time construction of suffix arrays. In: Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM), pp. 200–210 (2003)
23. Kufleitner, M.: On bijective variants of the Burrows–Wheeler transform. In: Proceedings of the Stringology, pp. 65–79 (2009)
24. Langmead, B., Trapnell, C., Pop, M., Salzberg, S.L.: Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol. **10**(3), R25 (2009)

25. Li, H., Durbin, R.: Fast and accurate short read alignment with Burrows–Wheeler transform. Bioinformatics **25**(14), 1754–1760 (2009)
26. Li, R., Yu, C., Li, Y., Lam, T.W., Yiu, S.M., Kristiansen, K., Wang, J.: Soap2: an improved ultrafast tool for short read alignment. Bioinformatics **25**(15), 1966–1967 (2009)
27. Lothaire, M.: Combinatorics on words. 2nd edn. Reading, MA (1983); Cambridge University Press, Cambridge (1997). Addison-Wesley (1983)
28. Lothaire, M.: Applied Combinatorics on Words (Encyclopedia of Mathematics and its Applications). Cambridge University Press, New York, NY (2005)
29. Lyndon, R.C.: On Burnside's problem. Trans. Am. Math. Soc. **77**, 202–215 (1954)
30. Lyndon, R.C.: On Burnside's problem II. Trans. Am. Math. Soc. **78**(2), 329–332 (1955)
31. Mantaci, S., Restivo, A., Rosone, G., Sciortino, M.: An extension of the Burrows– Wheeler transform and applications to sequence comparison and data compression. In: Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching (CPM), pp. 178–189 (2005)
32. Neuburger, S., Sokol, D.: Succinct 2D dictionary matching. Algorithmica **65**(3), 662–684 (2013)
33. Perret, L.: A chosen ciphertext attack on a public key cryptosystem based on Lyndon words. IACR Cryptol ePrint Arch **2005**, 14 (2005)
34. Reutenauer, C.: Free Lie Algebras. London Mathematical Society Monographs New Series. Oxford University Press, Oxford (1993)
35. Salson, M., Lecroq, T., Léonard, M., Mouchard, L.: A four-stage algorithm for updating a Burrows–Wheeler transform. Theor. Comput. Sci. **410**(43), 4350–4359 (2009)
36. Smyth, B.: Computing Patterns in Strings. ACM Press Bks, Addison-Wesley, Pearson (2003)
37. Tsai, Y.: The constrained longest common subsequence problem. Inf. Process. Lett. **88**(4), 173–176 (2003)
38. Wu, S., Manber, U.: Fast text searching: allowing errors. Commun. ACM **35**(10), 83–91 (1992)