A Text Transformation Scheme For Degenerate Strings

Jacqueline W. Daykin[1,2], Bruce Watson[1,3]

[1]Department of Informatics, King's College London, UK
[2]Department of Computer Science, Royal Holloway, University of London, UK
[3]Information Science Department, Stellenbosch University, South Africa
Jackie.Daykin@kcl.ac.uk bwwatson@sun.ac.za

Abstract

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. 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; we then 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 by applying lex-extension ordering. 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.

1 Introduction

This paper focuses on strings involving uncertainty – such strings are known as *indeterminate*, or equivalently, *degenerate* strings and consist of nonempty *subsets* of letters over an alphabet Σ[1] Algorithms for indeterminate strings have been described in [10].

Motivation for degenerate strings arises from applications such as interface data entry and bioinformatics. With degenerate biological strings, nucleotide sequences

In: Costas S. Iliopoulos, Alessio Langiu (eds.): Proceedings of the 2nd International Conference on Algorithms for Big Data, Palermo, Italy, 7-9 April 2014, published at http://ceur-ws.org/

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

are often written using the five letter alphabet $\{A, T, G, C, N\}$, where N denotes an unspecified nucleotide. For instance, $ANTAG$ may correspond to four different interpretations: $AATAG$, $ATTAG$, $AGTAG$ and $ACTAG$. Such degenerate strings can express polymorphisms in DNA/RNA sequences. Longest common subsequence computations apply to determining the homology of two biological sequences [20]; pattern matching techniques honed to degenerate DNA/RNA sequences are designed in [11].

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 [3].

In 1994, Burrows and Wheeler [2] introduced a transformation for textual data demonstrating, not only data clustering properties, but also suitability for block sorting compression. The Burrows-Wheeler Transform (BWT) operates by permuting the letters of a given 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 [13].

In Next-Generation Sequencing (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". Alignment programs attempt to align or match these reads to a reference genome; alignment was initially performed by applying hashing or the suffix tree/array data structures – subsequently, efficiency in memory requirement was achieved by using the BWT. Motivated by the degeneracy associated with genome sequencing, we introduce here a collection of novel and related concepts: a linear *Degenerate Burrows-Wheeler Transform*, an *Indeterminate Suffix Array*, *Indeterminate Conjugacy* and *Indeterminate Lyndon Words*.

2 Definitions and Preliminaries

A *string* (*word*) is a sequence of zero or more characters or letters over a totally ordered alphabet Σ. The set of all non-empty strings over Σ is denoted by Σ^+. The empty string is indicated by $\boldsymbol{\varepsilon}$; we write $\Sigma^* = \Sigma^+ \cup \boldsymbol{\varepsilon}$. Strings will be identified in mathbold such as $\boldsymbol{w}$, $\boldsymbol{x}$. We will use standard terminology from stringology: border, border-free, prefix, suffix, primitive, conjugate, etc. – see [19].

An *indeterminate* string $\boldsymbol{x} = \boldsymbol{x}[1 \dots m]$ on an alphabet Σ is a sequence of nonempty subsets of Σ; $\boldsymbol{x}$ is equivalently known as a *degenerate* string. Specifically, an indeterminate string $\boldsymbol{x}$ has the form $\boldsymbol{x} = \boldsymbol{x_1 x_2 \cdots x_m}$, where each $\boldsymbol{x_i}$ is a set of letters over Σ, 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 [2], and has become widely applied [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). 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 $\mathrm{BWT}(T) = (L, i)$. 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 [12], 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 [2]. Observe that the first column F of M can be obtained by lexicographically sorting the characters of L. By constructing a Hamiltonian cycle of L and F, the Last-First Mapping, the input can be recovered.

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; Kufleitner also proposed the bijective Sort Transform initiated earlier by Schindler – these variants are based on the Lyndon factorization of the input [9, 13, 18].

The BWT has also been implemented in bioinformatics: to reduce the memory requirement with hashing-based sequence alignment, BWT-based alignment utilities were developed including SOAP2 [15] and BOWTIE [14].

4 Indeterminate Lyndon Words

A *Lyndon word* is a primitive and border-free word which is strictly minimal for the lexicographical order of its conjugacy class [17] – let $\mathcal{L}$ denote the set of Lyndon words over the totally ordered alphabet Σ. These patterned words exhibit many interesting properties [16], including:

Proposition 4.1 *[8] A word $\boldsymbol{w} \in \Sigma^+$ is a Lyndon word if and only if it is lexicographically less than each of its nonempty proper suffixes.*

Proposition 4.2 *[8] 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, hence useful for applications.

Theorem 4.3 *[3] 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.*

In 1983, Duval [8] developed an algorithm for factorization that runs in linear time and constant space.

We now introduce the set $\mathcal{IL}$ of *Indeterminate Lyndon Words* – 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 $\mathcal{G} : \boldsymbol{x_i} \rightarrow \underline{\boldsymbol{x_i}}$ for $1 \leq i \leq m$, or simply $\mathcal{G} : \boldsymbol{x} \rightarrow \underline{\boldsymbol{x}}$. Furthermore, we can allow multisets under this mapping. Note that if $||\boldsymbol{x}|| = n$, and if we assume an integer alphabet, that is, if 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 sets $\boldsymbol{x_i}$ can be sorted in time $O(|\boldsymbol{x_i}|)$; hence the total time to compute $\underline{\boldsymbol{x}}$ is $O(n)$.

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

Definition [4, 6] Suppose that according to some factorization $\mathcal{F}$, two strings $\boldsymbol{u}, \boldsymbol{v} \in \Sigma^+$ are expressed in terms of nonempty factors:
$\boldsymbol{u} = \boldsymbol{u}_1\boldsymbol{u}_2 \cdots \boldsymbol{u}_m, \boldsymbol{v} = \boldsymbol{v}_1\boldsymbol{v}_2 \cdots \boldsymbol{v}_n$. Then $\boldsymbol{u} <_{LEX(\mathcal{F})} \boldsymbol{v}$ 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 $\boldsymbol{u} = \boldsymbol{u_1u_2} \cdots \boldsymbol{u_m}$, the factorization $\mathcal{F}$ is given by the sets $\boldsymbol{u_1u_2} \cdots \boldsymbol{u_m}$ mapped to $\underline{\boldsymbol{u_1u_2}} \cdots \underline{\boldsymbol{u_m}}$; for brevity we will write $\boldsymbol{u} <_{LEX} \boldsymbol{v}$. However, if all sets are unit size then the factorization $\mathcal{F}$ of regular strings is the individual letters, each $\underline{x_i}$ is x_i, and $\boldsymbol{u} <_{LEX} \boldsymbol{v}$ is simply the usual lexicographic order of strings $\boldsymbol{u} < \boldsymbol{v}$.

We can now proceed to clarify the concept of conjugacy for an indeterminate string.

Definition An indeterminate string $\boldsymbol{y} = \boldsymbol{y_1y_2} \cdots \boldsymbol{y_m}$ is a *conjugate* (or cyclic 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 An indeterminate string $\boldsymbol{x}$ over Σ^+ is an *Indeterminate Lyndon Word* if it is strictly minimal for the lex-extension order of its congugacy class under the mapping $\mathcal{G} : \boldsymbol{x} \rightarrow \underline{\boldsymbol{x}}$.

Similarly to each letter being a Lyndon word for regular strings, each single set of letters is likewise an indeterminate Lyndon word. Clearly Duval's linear Lyndon factorization algorithm [8] extends directly to the indeterminate case via lex-extension order and linear comparison of the substrings $\underline{\boldsymbol{x_i}}$. We can also trivially derive results analogous to those for the classic case – we give some examples, where $\mathcal{IL}$ is the set of Indeterminate Lyndon Words.

Proposition 4.4 *An indeterminate 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 nonempty proper suffixes.*

Proposition 4.5 *An indeterminate word $\boldsymbol{w} \in \Sigma^+$ is an indeterminate Lyndon word if and only if either $\boldsymbol{w}$ is a single set of letters or $\boldsymbol{w} = \boldsymbol{uv}$ with $\boldsymbol{u}$, $\boldsymbol{v} \in \mathcal{IL}$, $\boldsymbol{u} <_{LEX} \boldsymbol{v}$.*

A subset $\mathcal{W}$ of Σ^+ is known as a factorization family (FF) if and only if for every nonempty string $\boldsymbol{x}$ on Σ 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)* [4].

Lemma 4.6 *(The* **xyz** *Lemma [4]) 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 4.7 *[7] The set $\mathcal{IL}$ of Indeterminate Lyndon Words forms an UMFF.*

Furthermore, as detailed in [7] we are introducing here a new circ-UMFF [5], namely the set $\mathcal{IL}$ of *Indeterminate Lyndon Words*.

5 A Degenerate Burrows-Wheeler Transform

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. Given a degenerate string $\boldsymbol{x} = \boldsymbol{x}[1 \dots m] = \boldsymbol{x_1 x_2 \cdots x_m}$, to construct the D-BWT, we first perform all the mappings $\mathcal{G} : \boldsymbol{x_i} \rightarrow \underline{\boldsymbol{x_i}}$ specified in Section 4 in linear time. As in the original BWT transformation, we will generate the sorted rotations – the D-BWT matrix – of the input string. To do this we apply a fast suffix-sorting algorithm, such as that of Ko and Aluru [12], tweaked to handle substrings, thus forming an *indeterminate suffix array*. 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 $\{1, 2, \dots, n\}$.

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 π_i is just a letter or ordinal number. For example, $\underline{\boldsymbol{x}} = \{abc\}\{e\}\{ad\}\{abc\}\{bce\} \rightarrow ADBAC$ or 14213. This can be achieved using Bucket Sort on the finite ordered alphabet Σ (assumed in Section 4), with the buckets labelled by the characters in Σ. This process is repeated in each bucket where the length of each $\underline{\boldsymbol{x_i}}$ is $O(n)$ - hence $O(n)$ overall.

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 [12]) along with $O(n)$ total string length.

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

Furthermore, if we assume that the input text has been factored into indeterminate Lyndon words, then this avoids an index to the rotation in the matrix which is the input text. Once factored, and again using the re-labelling to letters $\underline{\boldsymbol{x_j}} \rightarrow \pi_i$, the bijective multi-word BWT described by Kufleitner [13] can be applied directly, followed by inverse mappings from the π_i to recover the indeterminate subsets in the input text.

References

[1] D. Adjeroh, T. Bell, and A. Mukherjee. *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching.* Springer Publishing Company, Incorporated, 1 edition, 2008.

[2] M. Burrows, D. J. Wheeler, M. Burrows, and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, 1994.

[3] K. T. Chen, R. H. Fox, and R. C. Lyndon. *Free differential calculus IV — The quotient groups of the lower central series*, volume 68. Ann. Math., 1958.

[4] D. E. Daykin and J. W. Daykin. Lyndon-like and v-order factorizations of strings. *J. Discrete Algorithms*, 1(3-4):357–365, 2003.

[5] D. E. Daykin and J. W. Daykin. Properties and construction of unique maximal factorization families for strings. *Int. J. Found. Comput. Sci.*, 19(4):1073–1084, 2008.

[6] J. W. Daykin and W. . F. Smyth. A bijective variant of the burrows-wheeler transform using v-order. 2013. Submitted.

[7] J. W. Daykin and B. Watson. Indeterminate string factorizations and degenerate text transformations. 2013. Submitted.

[8] J.-P. Duval. Factorizing words over an ordered alphabet. *J. Algorithms*, 4(4):363–381, 1983.

[9] J. Y. Gil and D. A. Scott. A bijective string sorting transform. *CoRR*, abs/1201.3077, 2012.

[10] J. Holub and W. F. Smyth. Algorithms on indeterminate strings. In *Proc. 14th Australasian Workshop on Combinatorial Algs.*, pages 36–45, 2003.

[11] C. S. Iliopoulos, L. Mouchard, and M. S. Rahman. A new approach to pattern matching in degenerate DNA/RNA sequences and distributed pattern matching. *Math. in Computer Science*, 2(4), 2008.

[12] P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. In *Proceedings of the 14th Annual Conference on Combinatorial Pattern Matching*, CPM'03, pages 200–210, Berlin, Heidelberg, 2003. Springer-Verlag.

[13] M. Kufleitner. On bijective variants of the Burrows-Wheeler Transform. In J. Holub and J. Zdárek, editors, *Stringology*, pages 65–79. Prague Stringology Club, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2009.

[14] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, 10(3):R25, 2009.

[15] R. Li, C. Yu, Y. Li, T. W. Lam, S. M. Yiu, K. Kristiansen, and J. Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[16] M. Lothaire. *Combinatorics on Words (Cambridge Mathematical Library).* Cambridge University Press; 2nd Edition, 1997.

[17] R. C. Lyndon. On Burnside's problem. *Transactions of the American Mathematical Society*, 77:202–215, 1954.

[18] M. Schindler. A fast block-sorting algorithm for lossless data compression. In *Proceedings of the Conference on Data Compression*, volume 469, 1997.

[19] W. Smyth. *Computing Patterns in Strings.* Addison-Wesley, 2003.

[20] Y.-T. Tsai. The constrained longest common subsequence problem. *Information Processing Letters*, 88(4):173 – 176, 2003.