On Bijective Variants of the Burrows-Wheeler Transform

Manfred Kufleitner

Universität Stuttgart, FMI,
Universitätsstr. 38, 70569 Stuttgart, Germany
kufleitner@fmi.uni-stuttgart.de

Abstract. The sort transform (ST) is a modification of the Burrows-Wheeler transform (BWT). Both transformations map an arbitrary word of length n to a pair consisting of a word of length n and an index between 1 and n. The BWT sorts all rotation conjugates of the input word, whereas the ST of order k only uses the first k letters for sorting all such conjugates. If two conjugates start with the same prefix of length k, then the indices of the rotations are used for tie-breaking. Both transforms output the sequence of the last letters of the sorted list and the index of the input within the sorted list. In this paper, we discuss a bijective variant of the BWT (due to Scott), proving its correctness and relations to other results due to Gessel and Reutenauer (1993) and Crochemore, Désarménien, and Perrin (2005). Further, we present a novel bijective variant of the ST.

1 Introduction

The Burrows-Wheeler transform (BWT) is a widely used preprocessing technique in lossless data compression [5]. It brings every word into a form which is likely to be easier to compress [18]. Its compression performance is almost as good as PPM (prediction by partial matching) schemes [7] while its speed is comparable to that of Lempel-Ziv algorithms [13,14]. Therefore, BWT based compression schemes are a very reasonable trade-off between running time and compression ratio.

In the classic setting, the BWT maps a word of length n to a word of length n and an index (comprising $O(\log n)$ bits). Thus, the BWT is not bijective and hence, it is introducing new redundancies to the data, which is cumbersome and undesired in applications of data compression or cryptography. Instead of using an index, a very common technique is to assume that the input has a unique end-of-string symbol [3,18]. Even though this often simplifies proofs or allows speeding up the algorithms, the use of an end-of-string symbol introduces new redundancies (again $O(\log n)$ bits are required for coding the end-of-string symbol).

We discuss bijective versions of the BWT which are one-to-one correspondences between words of length n. In particular, no index and no end-of-string symbol is needed. Not only does bijectivity save a few bits, for example, it also increases data security when cryptographic procedures are involved; it is more natural and it can help us to understand the BWT even better. Moreover, the bijective variants give us new possibilities for enhancements; for example, in the bijective BWT different orders on the letters can be used for the two main stages.

Several variants of the BWT have been introduced [2,17]. An overview can be found in the textbook by Adjeroh, Bell, and Mukherjee [1]. One particularly important variant for this paper is the sort transform (ST), which is also known under the name Schindler transform [22]. In the original paper, the inverse of the ST is described only

Manfred Kufleitner: On Bijective Variants of the Burrows-Wheeler Transform, pp. 65–79.
Proceedings of PSC 2009, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04403-2 © Czech Technical University in Prague, Czech Republic

very briefly. More precise descriptions and improved algorithms for the inverse of the ST have been proposed recently [19,20,21]. As for the BWT, the ST also involves an index or an end-of-string symbol. In particular, the ST is not onto and it introduces new redundancies.

The bijective BWT was discovered and first described by Scott (2007), but his exposition of the algorithm was somewhat cryptic, and was not appreciated as such. In particular, the fact that this transform is based on the Lyndon factorization went unnoticed by Scott. Gil and Scott [12] provided an accessible description of the algorithm. Here, we give an alternative description, a proof of its correctness, and more importantly, draw connections between Scott's algorithm and other results in combinatorics on words. Further, this variation of the BWT is used to introduce techniques which are employed at the bijective sort transform, which makes the main contribution of this paper. The forward transform of the bijective ST is rather easy, but we have to be very careful with some details. Compared with the inverse of the bijective BWT, the inverse of the bijective ST is more involved.

Outline. The paper is organized as follows. In Section 2 we fix some notation and repeat basic facts about combinatorics on words. On our way to the bijective sort transform (Section 6) we investigate the BWT (Section 3), the bijective BWT (Section 4), and the sort transform (Section 5). We give full constructive proofs for the injectivity of the respective transforms. Each section ends with a running example which illustrates the respective concepts. Apart from basic combinatorics on words, the paper is completely self-contained.

2 Preliminaries

Throughout this paper we fix the finite non-empty alphabet Σ and assume that Σ is equipped with a linear order $\leq$. A *word* is a sequence $a_1 \cdots a_n$ of letters $a_i \in \Sigma$, $1 \leq i \leq n$. The set of all such sequences is denoted by Σ^*; it is the free monoid over Σ with concatenation as composition and with the empty word ε as neutral element. The set $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ consists of all non-empty words. For words u, v we write $u \leq v$ if $u = v$ or if u is lexicographically smaller than v with respect to the order $\leq$ on the letters. Let $w = a_1 \cdots a_n \in \Sigma^+$ be a non-empty word with letters $a_i \in \Sigma$. The *length* of w, denoted by $|w|$, is n. The empty word is the unique word of length 0. We can think of w as a labeled linear order: position i of w is labeled by $a_i \in \Sigma$ and in this case we write $\lambda_w(i) = a_i$, so each word w induces a labeling function λ_w. The first letter a_1 of w is denoted by $\mathrm{first}(w)$ while the last letter a_n is denoted by $\mathrm{last}(w)$. The *reversal* of a word w is $\overline{w} = a_n \cdots a_1$. We say that two words u, v are *conjugate* if $u = st$ and $v = ts$ for some words s, t, i.e., u and v are cyclic shifts of one another. The j-fold concatenation of w with itself is denoted by w^j. A word u is a *root* of w if $w = u^j$ for some $j \in \mathbb{N}$. A word w is *primitive* if $w = u^j$ implies $j = 1$ and hence $u = w$, i.e., w has only the trivial root w.

The *right-shift* of $w = a_1 \cdots a_n$ is $r(w) = a_n a_1 \cdots a_{n-1}$ and the i-fold right shift $r^i(w)$ is defined inductively by $r^0(w) = w$ and $r^{i+1}(w) = r(r^i(w))$. We have $r^i(w) = a_{n-i+1} \cdots a_n a_1 \cdots a_{n-i}$ for $0 \leq i < n$. The word $r^i(w)$ is also well-defined for $i \geq n$ and then $r^i(w) = r^j(w)$ where $j = i \bmod n$. We define the *ordered conjugacy class* of a word $w \in \Sigma^n$ as $[w] = (w_1, \ldots, w_n)$ where $w_i = r^{i-1}(w)$. It is convenient to think of $[w]$ as a cycle of length n with a pointer to a distinguished

starting position. Every position i, $1 \leq i \leq n$, on this cycle is labeled by a_i. In particular, a_1 is a successor of a_n on this cycle since the position 1 is a successor of the position n. The mapping r moves the pointer to its predecessor. The (unordered) conjugacy class of w is the multiset $\{w_1, \ldots, w_n\}$. Whenever there is no confusion, then by abuse of notation we also write $[w]$ to denote the (unordered) conjugacy class of w. For instance, this is the case if w is in some way distinguished within its conjugacy class, which is true if w is a Lyndon word. A *Lyndon word* is a non-empty word which is the unique lexicographic minimal element within its conjugacy class. More formally, let $[w] = (w, w_2, \ldots, w_n)$, then $w \in \Sigma^+$ is a Lyndon word if $w < w_i$ for all $i \in \{2, \ldots, n\}$. Lyndon words have a lot of nice properties [15]. For instance, Lyndon words are primitive. Another interesting fact is the following.

Fact 1 (Chen, Fox, and Lyndon [6]). *Every word $w \in \Sigma^+$ has a unique factorization $w = v_s \cdots v_1$ such that $v_1 \leq \cdots \leq v_s$ is a non-decreasing sequence of Lyndon words.*

An alternative formulation of the above fact is that every word w has a unique factorization $w = v_s^{n_s} \cdots v_1^{n_1}$ where $n_i \geq 1$ for all i and where $v_1 < \cdots < v_s$ is a strictly increasing sequence of Lyndon words. The factorization of w as in Fact 1 is called the *Lyndon factorization* of w. It can be computed in linear time using Duval's algorithm [9].

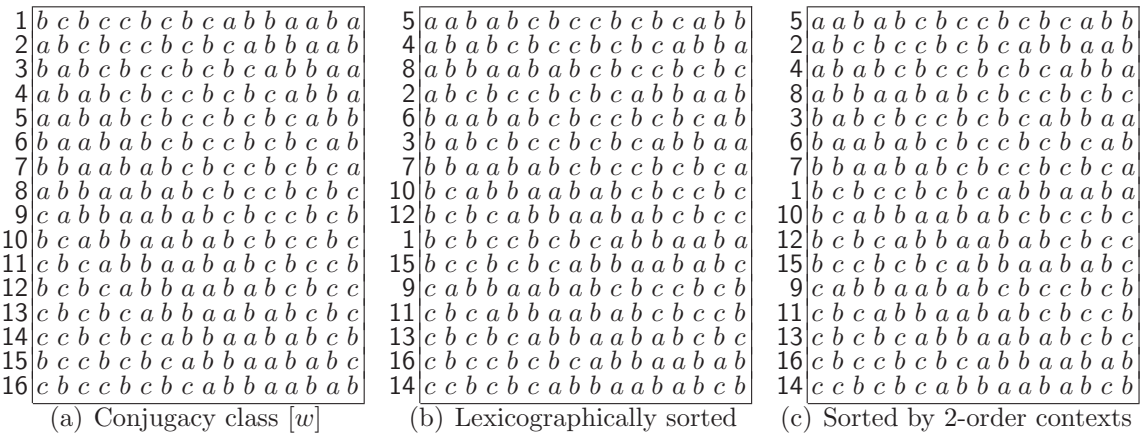
Suppose we are given a multiset $V = \{v_1, \ldots, v_s\}$ of Lyndon words enumerated in non-decreasing order $v_1 \leq \cdots \leq v_s$. Now, V uniquely determines the word $w = v_s \cdots v_1$. Therefore, the Lyndon factorization induces a one-to-one correspondence between arbitrary words of length n and multisets of Lyndon words of total length n. Of course, by definition of Lyndon words, the multiset $\{v_1, \ldots, v_s\}$ of Lyndon words and the multiset $\{[v_1], \ldots, [v_s]\}$ of conjugacy classes of Lyndon words are also in one-to-one correspondence.

We extend the order $\leq$ on Σ as follows to non-empty words. Let $w^\omega = www \cdots$ be the infinite sequences obtained as the infinite power of w. For $u, v \in \Sigma^+$ we write $u \leq^\omega v$ if either $u^\omega = v^\omega$ or $u^\omega = paq$ and $v^\omega = pbr$ for $p \in \Sigma^*$, $a, b \in \Sigma$ with $a < b$, and infinite sequences q, r; phrased differently, $u \leq^\omega v$ means that the infinite sequences u^ω and v^ω satisfy $u^\omega \leq v^\omega$. If u and v have the same length, then $\leq^\omega$ coincides with the lexicographic order induced by the order on the letters. For arbitrary words, $\leq^\omega$ is only a preorder since for example $u \leq^\omega uu$ and $uu \leq^\omega u$. On the other hand, if $u \leq^\omega v$ and $v \leq^\omega u$ then $u^{|v|} = v^{|u|}$. Hence, by the periodicity lemma [10], there exists a common root $p \in \Sigma^+$ and $g, h \in \mathbb{N}$ such that $u = p^g$ and $v = p^h$. Also note that $b \leq ba$ whereas $ba \leq^\omega b$ for $a < b$.

Intuitively, the *context of order k* of w is the sequence of the first k letters of w. We want this notion to be well-defined even if $|w| < k$. To this end let $\mathrm{context}_k(w)$ be the prefix of length k of w^ω, i.e., $\mathrm{context}_k(w)$ consists of the first k letters on the cycle $[w]$. Note that our definition of a context of order k is left-right symmetric to the corresponding notion used in data compression. This is due to the fact that typical compression schemes are applying the BWT or the ST to the reversal of the input.

An important construction in this paper is the *standard permutation* π_w on the set of positions $\{1, \ldots, n\}$ induced by a word $w = a_1 \cdots a_n \in \Sigma^n$ [11]. The first step is to introduce a new order $\preceq$ on the positions of w by sorting the letters within w such that identical letters preserve their order. More formally, the linear order $\preceq$ on $\{1, \ldots, n\}$ is defined as follows: $i \preceq j$ if

$$a_i < a_j \qquad \text{or} \qquad a_i = a_j \text{ and } i \leq j.$$



(a) Conjugacy class $[w]$ (b) Lexicographically sorted (c) Sorted by 2-order contexts

Figure 1. Computing the BWT and the ST of the word $w = bcbccbcbcabbaaba$

Let $j_1 \prec \cdots \prec j_n$ be the linearization of $\{1, \ldots, n\}$ according to this new order. Now, the standard permutation π_w is defined by $\pi_w(i) = j_i$.

Example 2. Consider the word $w = bcbccbcbcabbaaba$ over the ordered alphabet $a < b < c$. We have $|w| = 16$. Therefore, the positions in w are $\{1, \ldots, 16\}$. For instance, the label of position 6 is $\lambda_w(6) = b$. Its Lyndon factorization is $w = bcbcc \cdot bc \cdot bc \cdot abb \cdot aab \cdot a$. The context of order 7 of the prefix $bcbcc$ of length 5 is $bcbccbc$ and the context of order 7 of the factor bc is $bcbcbcb$. For computing the standard permutation we write w column-wise, add positions, and then sort the pairs lexicographically:

word w	w with positions	sorted
b	$(b,1)$	$(a,10)$
c	$(c,2)$	$(a,13)$
b	$(b,3)$	$(a,14)$
c	$(c,4)$	$(a,16)$
c	$(c,5)$	$(b,1)$
b	$(b,6)$	$(b,3)$
c	$(c,7)$	$(b,6)$
b	$(b,8)$	$(b,8)$
c	$(c,9)$	$(b,11)$
a	$(a,10)$	$(b,12)$
b	$(b,11)$	$(b,15)$
b	$(b,12)$	$(c,2)$
a	$(a,13)$	$(c,4)$
a	$(a,14)$	$(c,5)$
b	$(b,15)$	$(c,7)$
a	$(a,16)$	$(c,9)$

This yields the standard permutation

$$\pi_w = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 10 & 13 & 14 & 16 & 1 & 3 & 6 & 8 & 11 & 12 & 15 & 2 & 4 & 5 & 7 & 9 \end{pmatrix}.$$

The conjugacy class $[w]$ of w is depicted in Figure 1(a); the i-th word in $[w]$ is written in the i-th row. The last column of the matrix for $[w]$ is the reversal $\overline{w}$ of w.

3 The Burrows-Wheeler transform

The *Burrows-Wheeler transform* (BWT) maps words w of length n to pairs (L, i) where L is a word of length n and i is an index in $\{1, \ldots, n\}$. The word L is usually

referred to as the Burrows-Wheeler transform of w. In particular, the BWT is not surjective. We will see below how the BWT works and that it is one-to-one. It follows that only a fraction of $1/n$ of all possible pairs (L, i) appears as an image under the BWT. For instance $(bacd, 1)$ where $a < b < c < d$ is not an image under the BWT.

For $w \in \Sigma^+$ we define $M(w) = (w_1, \ldots, w_n)$ where $\{w_1, \ldots, w_n\} = [w]$ and $w_1 \leq \cdots \leq w_n$. Now, the Burrows-Wheeler transform of w consists of the word $\mathrm{BWT}(w) = \mathrm{last}(w_1) \cdots \mathrm{last}(w_n)$ and an index i such that $w = w_i$. Note that in contrast to the usual definition of the BWT, we are using right shifts; at this point this makes no difference but it unifies the presentation of succeeding transforms. At first glance, it is surprising that one can reconstruct $M(w)$ from $\mathrm{BWT}(w)$. Moreover, if we know the index i of w in the sorted list $M(w)$, then we can reconstruct w from $\mathrm{BWT}(w)$. One way of how to reconstruct $M(w)$ is presented in the following lemma. For later use, we prove a more general statement than needed for computing the inverse of the BWT.

Lemma 3. *Let $k \in \mathbb{N}$. Let $\bigcup_{i=1}^{s} [v_i] = \{w_1, \ldots, w_n\} \subseteq \Sigma^+$ be a multiset built from conjugacy classes $[v_i]$. Let $M = (w_1, \ldots, w_n)$ satisfy $\mathrm{context}_k(w_1) \leq \cdots \leq \mathrm{context}_k(w_n)$ and let $L = \mathrm{last}(w_1) \cdots \mathrm{last}(w_n)$ be the sequence of the last symbols. Then*

$$\mathrm{context}_k(w_i) = \lambda_L \pi_L(i) \cdot \lambda_L \pi_L^2(i) \cdots \lambda_L \pi_L^k(i)$$

where π_L^t denotes the t-fold application of π_L and $\lambda_L \pi_L^t(i) = \lambda_L\big(\pi_L^t(i)\big)$.

Proof. By induction over the context length t, we prove that for all $i \in \{1, \ldots, n\}$ we have $\mathrm{context}_t(w_i) = \lambda_L \pi_L(i) \cdots \lambda_L \pi_L^t(i)$. For $t = 0$ we have $\mathrm{context}_0(w_i) = \varepsilon$ and hence, the claim is trivially true. Let now $0 < t \leq k$. By the induction hypothesis, the $(t-1)$-order context of each w_i is $\lambda_L \pi_L(i) \cdots \lambda_L \pi_L^{t-1}(i)$. By applying one right-shift, we see that the t-order context of $r(w_i)$ is $\lambda_L(i) \cdot \lambda_L \pi_L^1(i) \cdots \lambda_L \pi_L^{t-1}(i)$.

The list M meets the sort order induced by k-order contexts. In particular, $(w_1, \ldots, w_n)$ is sorted by $(t-1)$-order contexts. Let $(u_1, \ldots, u_n)$ be a stable sort by t-order contexts of the right-shifts $(r(w_1), \ldots, r(w_n))$. The construction of $(u_1, \ldots, u_n)$ only requires a sorting of the first letters of $(r(w_1), \ldots, r(w_n))$ such that identical letters preserve their order. The sequence of first letters of the words $r(w_1), \ldots, r(w_n)$ is exactly L. By construction of π_L, it follows that $(u_1, \ldots, u_n) = (w_{\pi_L(1)}, \ldots, w_{\pi_L(n)})$. Since M is built from conjugacy classes, the multisets of elements occurring in $(w_1, \ldots, w_n)$ and $(r(w_1), \ldots, r(w_n))$ are identical. The same holds for the multisets induced by $(w_1, \ldots, w_n)$ and $(u_1, \ldots, u_n)$. Therefore, the sequences of t-order contexts induced by $(w_1, \ldots w_n)$ and $(u_1, \ldots, u_n)$ are identical. Moreover, we conclude

$$\mathrm{context}_t(w_i) = \mathrm{context}_t(u_i) = \mathrm{context}_t(w_{\pi_L(i)}) = \lambda_L \pi_L(i) \cdot \lambda_L \pi_L^2(i) \cdots \lambda_L \pi_L^t(i)$$

which completes the induction. We note that in general $u_i \neq w_i$ since the sort order of M beyond k-order contexts is arbitrary. Moreover, for $t = k + 1$ the property $\mathrm{context}_t(w_i) = \mathrm{context}_t(u_i)$ does not need to hold (even though the multisets of $(k+1)$-order contexts coincide). □

Note that in Lemma 3 we do not require that all v_i have the same length. Applying the BWT to conjugacy classes of words with different lengths has also been used for the *Extended BWT* [17].

Corollary 4. *The BWT is invertible, i.e., given $(\mathrm{BWT}(w), i)$ where i is the index of w in $M(w)$ one can reconstruct the word w.*

Proof. We set $k = |w|$. Let $M = M(w)$ and $L = \mathrm{BWT}(w)$. Now, by Lemma 3 we see that

$$w = w_i = \mathrm{context}_k(w_i) = \lambda_L \pi_L^1(i) \cdots \lambda_L \pi_L^{|L|}(i).$$

In particular, $w = \lambda_L \pi_L^1(i) \cdots \lambda_L \pi_L^{|L|}(i)$ only depends on L and i. □

Remark 5. In the special case of the BWT it is possible to compute the i-th element w_i of $M(w)$ by using the inverse π_L^{-1} of the permutation π_L:

$$w_i = \lambda_L \pi_L^{-|w_i|+1}(i) \cdots \lambda_L \pi_L^{-1}(i) \lambda_L(i).$$

This justifies the usual way of computing the inverse of $(\mathrm{BWT}(w), i)$ from right to left (by using the restriction of π_L^{-1} to the cycle containing the element i). The motivation is that the (required cycle of the) inverse π_L^{-1} seems to be easier to compute than the standard permutation π_L.

Example 6. We compute the BWT of $w = bcbccbcbcabbaaba$ from Example 2. The lexicographically sorted list $M(w)$ can be found in Figure 1(b). This yields the transform $(\mathrm{BWT}(w), i) = (bacbbaaccacbbcbb, 10)$ where $L = \mathrm{BWT}(w)$ is the last column of the matrix $M(w)$ and w is the i-th row in $M(w)$. The standard permutation of L is

$$\pi_L = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 2 & 6 & 7 & 10 & 1 & 4 & 5 & 12 & 13 & 15 & 16 & 3 & 8 & 9 & 11 & 14 \end{pmatrix}.$$

Now, $\pi_L^1(10) \cdots \pi_L^{16}(10)$ gives us the following sequence of positions starting with $\pi_L(10) = 15$:

$$15 \overset{\pi_L}{\mapsto} 11 \overset{\pi_L}{\mapsto} 16 \overset{\pi_L}{\mapsto} 14 \overset{\pi_L}{\mapsto} 9 \overset{\pi_L}{\mapsto} 13 \overset{\pi_L}{\mapsto} 8 \overset{\pi_L}{\mapsto} 12 \overset{\pi_L}{\mapsto} 3 \overset{\pi_L}{\mapsto} 7 \overset{\pi_L}{\mapsto} 5 \overset{\pi_L}{\mapsto} 1 \overset{\pi_L}{\mapsto} 2 \overset{\pi_L}{\mapsto} 6 \overset{\pi_L}{\mapsto} 4 \overset{\pi_L}{\mapsto} 10.$$

Applying the labeling function λ_L to this sequence of positions yields

$$\begin{aligned} &\lambda_L(15)\lambda_L(11)\lambda_L(16)\lambda_L(14)\lambda_L(9)\lambda_L(13)\lambda_L(8)\lambda_L(12) \\ &\quad \cdot \lambda_L(3)\lambda_L(7)\lambda_L(5)\lambda_L(1)\lambda_L(2)\lambda_L(6)\lambda_L(4)\lambda_L(10) \\ &= bcbccbcbcabbaaba = w, \end{aligned}$$

i.e., we have successfully reconstructed the input w from $(\mathrm{BWT}(w), i)$.

4 The bijective Burrows-Wheeler transform

Now we are ready to give a comprehensive description of Scott's bijective variant of the BWT and to prove its correctness. It maps a word of length n to a word of length n — without any index or end-of-string symbol being involved. The key ingredient is the Lyndon factorization: Suppose we are computing the BWT of a Lyndon word v, then we do not need an index since we know that v is the first element of the list $M(v)$. This leads to the computation of a multi-word BWT of the Lyndon factors of the input.

The bijective BWT of a word w of length n is defined as follows. Let $w = v_s \cdots v_1$ with $v_s \geq \cdots \geq v_1$ be the Lyndon factorization of w. Let $\mathrm{LM}(w) = (u_1, \ldots, u_n)$ where $u_1 \leq^\omega \cdots \leq^\omega u_n$ and where the multiset $\{u_1, \ldots, u_n\} = \bigcup_{i=1}^s [v_i]$. Then, the bijective BWT of w is $\mathrm{BWTS}(w) = \mathrm{last}(u_1) \cdots \mathrm{last}(u_n)$. The S in BWTS is for *Scottified*. Note that if w is a power of a Lyndon word, then $\mathrm{BWTS}(w) = \mathrm{BWT}(w)$.

In some sense, the bijective BWT can be thought of as the composition of the Lyndon factorization [6] with the inverse of the Gessel-Reutenauer transform [11]. In particular, a first step towards a bijective BWT can be found in a 1993 article by Gessel and Reutenauer [11] (prior to the publication of the BWT [5]). The link between the Gessel-Reutenauer transform and the BWT was pointed out later by Crochemore et al. [8]. A similar approach as in the bijective BWT has been employed by Mantaci et al. [16]; instead of the Lyndon factorization they used a decomposition of the input into blocks of equal length. The output of this variant is a word and a sequence of indices (one for each block). In its current form, the bijective BWT has been proposed by Scott [23] in a newsgroup posting in 2007. Gil and Scott gave an accessible version of the transform, an independent proof of its correctness, and they tested its performance in data compression [12]. The outcome of these tests is that the bijective BWT beats the usual BWT on almost all files of the Calgary Corpus [4] by at least a few hundred bytes which exceeds the gain of just saving the rotation index.

Lemma 7. *Let $w = v_s \cdots v_1$ with $v_s \geq \cdots \geq v_1$ be the Lyndon factorization of w, let $\mathrm{LM}(w) = (u_1, \ldots, u_n)$, and let $L = \mathrm{BWTS}(w)$. Consider the cycle C of the permutation π_L which contains the element 1 and let d be the length of C. Then $\lambda_L \pi_L^1(1) \cdots \lambda_L \pi_L^d(1) = v_1$.*

Proof. By Lemma 3 we see that $\big(\lambda_L \pi_L^1(1) \cdots \lambda_L \pi_L^d(1)\big)^{|v_1|} = v_1^d$. Since v_1 is primitive it follows $\lambda_L \pi_L^1(1) \cdots \lambda_L \pi_L^d(1) = v_1^z$ for some $z \in \mathbb{N}$. In particular, the Lyndon factorization of w ends with v_1^z.

Let U be the subsequence of $\mathrm{LM}(w)$ which consists of those u_i which come from this last factor v_1^z. The sequence U contains each right-shift of v_1 exactly z times. Moreover, the sort-order within U depends only on $|v_1|$-order contexts.

The element $v_1 = u_1$ is the first element in U since v_1 is a Lyndon word. In particular, $\pi_L^0(1) = 1$ is the first occurrence of $r^0(v_1) = v_1$ within U. Suppose $\pi_L^j(1)$ is the first occurrence of $r^j(v_1)$ within U. Let $\pi_L^j(1) = i_1 < \cdots < i_z$ be the indices of all occurrences of $r^j(v_1)$ in U. By construction of π_L, we have $\pi_L(i_1) < \cdots < \pi_L(i_z)$ and therefore $\pi_L^{j+1}(1)$ is the first occurrence of $r^{j+1}(v_1)$ within U. Inductively, $\pi_L^j(1)$ always refers to the first occurrence of $r^j(v_1)$ within U (for all $j \in \mathbb{N}$). In particular it follows that $\pi_L^{|v_1|}(1) = 1$ and $z = 1$. □

Theorem 8. *The bijective BWT is invertible, i.e., given $\mathrm{BWTS}(w)$ one can reconstruct the word w.*

Proof. Let $L = \mathrm{BWTS}(w)$ and let $w = v_s \cdots v_1$ with $v_s \geq \cdots \geq v_1$ be the Lyndon factorization of w. Each permutation admits a cycle structure. We decompose the standard permutation π_L into cycles $C_1, \ldots, C_t$. Let i_j be the smallest element of the cycle C_j and let d_j be the length of C_j. We can assume that $1 = i_1 < \cdots < i_t$.

We claim that $t = s$, $d_j = |v_j|$, and $\lambda_L \pi_L^1(i_j) \cdots \lambda_L \pi_L^{d_j}(i_j) = v_j$. By Lemma 7 we have $\lambda_L \pi_L^1(i_1) \cdots \lambda_L \pi_L^{d_1}(i_1) = v_1$. Let π'_L denote the restriction of π_L to the set $C = C_2 \cup \cdots \cup C_t$, where by abuse of notation $C_2 \cup \cdots \cup C_t$ denotes the set of all elements occurring in $C_2, \ldots, C_t$. Let $L' = \mathrm{BWTS}(v_s \cdots v_2)$. The word L' can be obtained from L by removing all positions occurring in the cycle C_1. This yields a monotone bijection

$$\alpha : C \to \{1, \ldots, |L'|\}$$

such that $\lambda_L(i) = \lambda_{L'}\alpha(i)$ and $\alpha\pi_L(i) = \pi_{L'}\alpha(i)$ for all $i \in C$. In particular, $\pi_{L'}$ has the same cycle structure as π'_L and $1 = \alpha(i_2) < \cdots < \alpha(i_t)$ is the sequence of the minimal elements within the cycles. By induction on the number of Lyndon factors,

$$\begin{aligned} v_s \cdots v_2 &= \lambda_{L'}\pi^1_{L'}\alpha(i_t) \cdots \lambda_{L'}\pi^{d_t}_{L'}\alpha(i_t) \ \cdots \ \lambda_{L'}\pi^1_{L'}\alpha(i_2) \cdots \lambda_{L'}\pi^{d_2}_{L'}(i_2) \\ &= \lambda_{L'}\alpha\pi^1_{L}(i_t) \cdots \lambda_{L'}\alpha\pi^{d_t}_{L}(i_t) \ \cdots \ \lambda_{L'}\alpha\pi^1_{L}(i_2) \cdots \lambda_{L'}\alpha\pi^{d_2}_{L}(i_2) \\ &= \lambda_{L}\pi^1_{L}(i_t) \cdots \lambda_{L}\pi^{d_t}_{L}(i_t) \ \cdots \ \lambda_{L}\pi^1_{L}(i_2) \cdots \lambda_{L}\pi^{d_2}_{L}(i_2). \end{aligned}$$

Appending $\lambda_L\pi^1_L(i_1) \cdots \lambda_L\pi^{d_1}_L(i_1) = v_1$ to the last line allows us to reconstruct w by

$$w = \lambda_L\pi^1_L(i_t) \cdots \lambda_L\pi^{d_t}_L(i_t) \ \cdots \ \lambda_L\pi^1_L(i_1) \cdots \lambda_L\pi^{d_1}_L(i_1).$$

Moreover, $t = s$ and $d_j = |v_j|$. We note that this formula for w only depends on L and does not require any index to an element in $\mathrm{LM}(w)$. □

Example 9. We again consider the word $w = bcbccbcbcabbaaba$ from Example 2 and its Lyndon factorization $w = v_6 \cdots v_1$ where $v_6 = bcbcc$, $v_5 = bc$, $v_4 = bc$, $v_3 = abb$, $v_2 = aab$, and $v_1 = a$. The lists $([v_1], \ldots, [v_6])$ and $\mathrm{LM}(w)$ are:

	$([v_1], \ldots, [v_6])$
1	a
2	$a\ a\ b$
3	$b\ a\ a$
4	$a\ b\ a$
5	$a\ b\ b$
6	$b\ a\ b$
7	$b\ b\ a$
8	$b\ c$
9	$c\ b$
10	$b\ c$
11	$c\ b$
12	$b\ c\ b\ c\ c$
13	$c\ b\ c\ b\ c$
14	$c\ c\ b\ c\ b$
15	$b\ c\ c\ b\ c$
16	$c\ b\ c\ c\ b$

	$\mathrm{LM}(w)$
1	a
2	$a\ a\ b$
4	$a\ b\ a$
5	$a\ b\ b$
3	$b\ a\ a$
6	$b\ a\ b$
7	$b\ b\ a$
8	$b\ c$
10	$b\ c$
12	$b\ c\ b\ c\ c$
15	$b\ c\ c\ b\ c$
9	$c\ b$
11	$c\ b$
13	$c\ b\ c\ b\ c$
16	$c\ b\ c\ c\ b$
14	$c\ c\ b\ c\ b$

Hence, we obtain $L = \mathrm{BWTS}(w) = abababaccccbbcbb$ as the sequence of the last symbols of the words in $\mathrm{LM}(w)$. The standard permutation π_L induced by L is

$$\pi_L = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 1 & 3 & 5 & 7 & 2 & 4 & 6 & 12 & 13 & 15 & 16 & 8 & 9 & 10 & 11 & 14 \end{pmatrix}$$

The cycles of π_L arranged by their smallest elements are $C_1 = (1)$, $C_2 = (2, 3, 5)$, $C_3 = (4, 7, 6)$, $C_4 = (8, 12)$, $C_5 = (9, 13)$, and $C_6 = (10, 15, 11, 16, 14)$. Applying the labeling function λ_L to the cycle C_i (starting with the second element) yields the Lyndon factor v_i. With this procedure, we reconstructed $w = v_6 \cdots v_1$ from $L = \mathrm{BWTS}(w)$.

5 The sort transform

The *sort transform* (ST) is a BWT where we only sort the conjugates of the input up to a given depth k and then we are using the index of the conjugates as a tie-breaker. Depending on the depth k and the implementation details this can speed up compression (while at the same time slightly slowing down decompression).

In contrast to the usual presentation of the ST, we are using right shifts. This defines a slightly different version of the ST. The effect is that the order of the symbols occurring in some particular context is reversed. This makes sense, because in data compression the ST is applied to the reversal of a word. Hence, in the ST of the reversal of w the order of the symbols in some particular context is the same as in w. More formally, suppose $\overline{w} = x_0 c a_1 x_1 c a_2 x_2 \cdots c a_s x_s$ for $c \in \Sigma^+$ then in the sort transform of order $|c|$ of w, the order of the occurrences of the letters a_i is not changed. This property can enable better compression ratios on certain data.

While the standard permutation is induced by a sequence of letters (i.e., a word) we now generalize this concept to sequences of words. For a list of non-empty words $V = (v_1, \ldots, v_n)$ we now define the *k-order standard permutation* $\nu_{k,V}$ induced by V. As for the standard permutation, the first step is the construction of a new linear order $\preceq$ on $\{1, \ldots, n\}$. We define $i \preceq j$ by the condition

$$\mathrm{context}_k(v_i) < \mathrm{context}_k(v_j) \qquad \text{or} \qquad \mathrm{context}_k(v_i) = \mathrm{context}_k(v_j) \text{ and } i \leq j.$$

Let $j_1 \prec \cdots \prec j_n$ be the linearization of $\{1, \ldots, n\}$ according to this new order. The idea is that we sort the line numbers of $v_1, \ldots, v_n$ by first considering the k-order contexts and, if these are equal, then use the line numbers as tie-breaker. As before, the linearization according to $\preceq$ induces a permutation $\nu_{k,V}$ by setting $\nu_{k,V}(i) = j_i$. Now, $\nu_{k,V}(i)$ is the position of v_i if we are sorting V by k-order context such that the line numbers serve as tie-breaker. We set $M_k(v_1, \ldots, v_n) = (w_1, \ldots, w_n)$ where $w_i = v_{\nu_{k,V}(i)}$. Now, we are ready to define the sort transform of order k of a word w: Let $M_k([w]) = (w_1, \ldots, w_n)$; then $\mathrm{ST}_k(w) = \mathrm{last}(w_1) \cdots \mathrm{last}(w_n)$, i.e., we first sort all cyclic right-shifts of w by their k-order contexts (by using a stable sort method) and then we take the sequence of last symbols according to this new sort order as the image under ST_k. Since the tie-breaker relies on right-shifts, we have $\mathrm{ST}_0(w) = \overline{w}$, i.e., ST_0 is the reversal mapping. The k-order sort transform of w is the pair $(\mathrm{ST}_k(w), i)$ where i is the index of w in $M_k([w])$. As for the BWT, we see that the k-order sort transform is not bijective.

Next, we show that it is possible to reconstruct $M_k([w])$ from $\mathrm{ST}_k(w)$. Hence, it is possible to reconstruct w from the pair $(\mathrm{ST}_k(w), i)$ where i is the index of w in $M_k([w])$. The presentation of the back transform is as follows. First, we will introduce the *k-order context graph* G_k and we will show that it is possible to rebuild $M_k([w])$ from G_k. Then we will show how to construct G_k from $\mathrm{ST}_k(w)$. Again, the approach will be slightly more general than required at the moment; but we will be able to reuse it in the presentation of a bijective ST.

Let $V = ([u_1], \ldots, [u_s]) = (v_1, \ldots, v_n)$ be a list of words built from conjugacy classes $[u_i]$ of non-empty words u_i. Let $M = (w_1, \ldots, w_n)$ be an arbitrary permutation of the elements in V. We are now describing the edge-labeled directed graph $G_k(M)$ – the k-order context graph of M – which will be used later as a presentation tool for the inverses of the ST and the bijective ST. The vertices of $G_k(M)$ consist of all k-order contexts $\mathrm{context}_k(w)$ of words w occurring in M. We draw an edge (c_1, i, c_2) from context c_1 to context c_2 labeled by i if $c_1 = \mathrm{context}_k(w_i)$ and $c_2 = \mathrm{context}_k(r(w_i))$. Hence, every index $i \in \{1, \ldots, n\}$ of M defines a unique edge in $G_k(M)$. We can also think of $\mathrm{last}(w_i)$ as an additional implicit label of the edge (c_1, i, c_2), since $c_2 = \mathrm{context}_k(\mathrm{last}(w_i)c_1)$.

A *configuration* $(\mathcal{C}, c)$ of the k-order context graph $G_k(M)$ consists of a subset of the edges $\mathcal{C}$ and a vertex c. The idea is that (starting at context c) we are walking

along the edges of $G_k(M)$ and whenever an edge is used, it is removed from the set of edges $\mathcal{C}$. We now define the transition

$$(\mathcal{C}_1, c_1) \xrightarrow{u} (\mathcal{C}_2, c_2)$$

from a configuration $(\mathcal{C}_1, c_1)$ to another configuration $(\mathcal{C}_2, c_2)$ with output $u \in \Sigma^*$ more formally. If there exists an edge in $\mathcal{C}_1$ starting at c_1 and if $(c_1, i, c_2) \in \mathcal{C}_1$ is the unique edge with the smallest label i starting at c_1, then we have the single-step transition

$$(\mathcal{C}_1, c_1) \xrightarrow{a} (\mathcal{C}_1 \setminus \{(c_1, i, c_2)\}, c_2) \qquad \text{where } a = \text{last}(w_i)$$

If there is no edge in $\mathcal{C}_1$ starting at c_1, then the outcome of $(\mathcal{C}_1, c_1) \rightarrow$ is undefined. Inductively, we define $(\mathcal{C}_1, c_1) \xrightarrow{\varepsilon} (\mathcal{C}_1, c_1)$ and for $a \in \Sigma$ and $u \in \Sigma^*$ we have

$$(\mathcal{C}_1, c_1) \xrightarrow{au} (\mathcal{C}_2, c_2) \quad \text{if} \quad (\mathcal{C}_1, c_1) \xrightarrow{u} (\mathcal{C}', c') \text{ and } (\mathcal{C}', c') \xrightarrow{a} (\mathcal{C}_2, c_2)$$

for some configuration $(\mathcal{C}', c')$. Hence, the reversal $\overline{au}$ is the label along the path of length $|au|$ starting at configuration $(\mathcal{C}_1, c_1)$. In particular, if $(\mathcal{C}_1, c_1) \xrightarrow{u} (\mathcal{C}_2, c_2)$ holds, then it is possible to chase at least $|u|$ transitions starting at $(\mathcal{C}_1, c_1)$; vice versa, if we are chasing ℓ transitions then we obtain a word of length ℓ as a label. We note that successively taking the edge with the smallest label comes from the use of right-shifts. If we had used left-shifts we would have needed to chase largest edges for the following lemma to hold. The reverse labeling of the big-step transitions is motivated by the reconstruction procedure which will work from right to left.

Lemma 10. *Let $k \in \mathbb{N}$, $V = ([v_1], \ldots, [v_s])$, $c_i = \text{context}_k(v_i)$, and $G = G_k(M_k(V))$. Let $\mathcal{C}_1$ consist of all edges of G. Then*

$$\begin{aligned}
(\mathcal{C}_1, c_1) &\xrightarrow{v_1} (\mathcal{C}_2, c_1) \\
(\mathcal{C}_2, c_2) &\xrightarrow{v_2} (\mathcal{C}_3, c_2) \\
&\vdots \\
(\mathcal{C}_s, c_s) &\xrightarrow{v_s} (\mathcal{C}_{s+1}, c_s).
\end{aligned}$$

Proof. Let $M_k(V) = (w_1, \ldots, w_n)$. Consider some index i, $1 \le i \le s$, and let $(u_1, \ldots, u_t) = ([v_1], \ldots, [v_{i-1}])$. Suppose that $\mathcal{C}_i$ consists of all edges of G except for those with labels $\nu_{k,V}(j)$ for $1 \le j \le t$. Let $q = |v_i|$. We write $v_i = a_1 \cdots a_q$ and $u_{t+j} = r^{j-1}(v_i)$, i.e., $[v_i] = (u_{t+1}, \ldots, u_{t+q})$. Starting with $(\mathcal{C}_{i,1}, c_{i,1}) = (\mathcal{C}_i, c_i)$, we show that the sequence of transitions

$$(\mathcal{C}_{i,1}, c_{i,1}) \xrightarrow{a_q} (\mathcal{C}_{i,2}, c_{i,2}) \xrightarrow{a_{q-1}} \cdots (\mathcal{C}_{i,q}, c_{i,q}) \xrightarrow{a_1} (\mathcal{C}_{i,q+1}, c_{i,q+1})$$

is defined. More precisely, we will see that the transition $(\mathcal{C}_{i,j}, c_{i,j}) \xrightarrow{a_{q+1-j}} (\mathcal{C}_{i,j+1}, c_{i,j+1})$ walks along the edge $\big(c_{i,j}, \nu_{k,V}(t+j), c_{i,j+1}\big)$ and hence indeed is labeled with the letter $a_{q+1-j} = \text{last}(u_{t+j}) = \text{last}(w_{\nu_{k,V}(t+j)})$. Consider the context $c_{i,j}$. By induction, we have $c_{i,j} = \text{context}_k(u_{t+j})$ and no edge with label $\nu_{k,V}(\ell)$ for $1 \le \ell < t+j$ occurs in $\mathcal{C}_{i,j}$ while all other labels do occur. In particular, $(c_{i,j}, \nu_{k,V}(t+j), c_{i,j+1})$ for $c_{i,j+1} = \text{context}_k(r(u_{t+j})) = \text{context}_k(u_{t+j+1})$ is an edge in $\mathcal{C}_{i,j}$ (where $\text{context}_k(r(u_{t+j})) = \text{context}_k(u_{t+j+1})$ only holds for $j < q$; we will consider the case $j = q$ below). Suppose there were an edge $(c_{i,j}, z, c') \in \mathcal{C}_{i,j}$ with $z < \nu_{k,V}(t+j)$. Then $\text{context}_k(w_z) = c_{i,j}$ and hence, w_z has the same k-order context as $w_{\nu_{k,V}(t+j)}$. But in this case, in the construction of $M_k(V)$ we used the index in V as a tie-breaker. It follows $\nu^{-1}_{k,V}(z) < t+1$ which

contradicts the properties of $\mathcal{C}_{i,j}$. Hence, $(c_{i,j}, \nu_{k,V}(t+j), c_{i,j+1})$ is the edge with the smallest label starting at context $c_{i,j}$. Therefore, $\mathcal{C}_{i,j+1} = \mathcal{C}_{i,j} \backslash \{(c_{i,j}, \nu_{k,V}(t+j), c_{i,j+1})\}$ and $(\mathcal{C}_{i,j}, c_{i,j}) \xrightarrow{a_{q+1-j}} (\mathcal{C}_{i,j+1}, c_{i,j+1})$ indeed walks along the edge $(c_{i,j}, \nu_{k,V}(t+j), c_{i,j+1})$.

It remains to verify that $c_{i,1} = c_{i,q+1}$, but this is clear since $c_{i,1} = \mathrm{context}_k(u_{t+1}) = \mathrm{context}_k(r^q(u_{t+1})) = c_{i,q+1}$. □

Lemma 11. *Let $k \in \mathbb{N}$, $V = ([v_1], \ldots, [v_s])$, $M = M_k(V) = (w_1, \ldots, w_n)$, and $L = \mathrm{last}(w_1) \cdots \mathrm{last}(w_n)$. Then it is possible to reconstruct $G_k(M)$ from L.*

Proof. By Lemma 3 it is possible to reconstruct the contexts $c_i = \mathrm{context}_k(w_i)$. This gives the vertices of the graph $G_k(M)$. Write $L = a_1 \cdots a_n$. For each $i \in \{1, \ldots, n\}$ we draw an edge $(c_i, i, \mathrm{context}_k(a_i c_i))$. This yields the edges of $G_k(M)$. □

Corollary 12. *The k-order ST is invertible, i.e., given $(\mathrm{ST}_k(w), i)$ where i is the index of w in $M_k([w])$ one can reconstruct the word w.*

Proof. The construction of w consists of two phases. First, by Lemma 11 we can compute $G_k(M_k([w]))$. By Lemma 3 we can compute $c = \mathrm{context}_k(w)$ from $(\mathrm{ST}_k(w), i)$. In the second stage, we are using Lemma 10 for reconstructing w by chasing

$$(\mathcal{C}, c) \xrightarrow{w} (\emptyset, c)$$

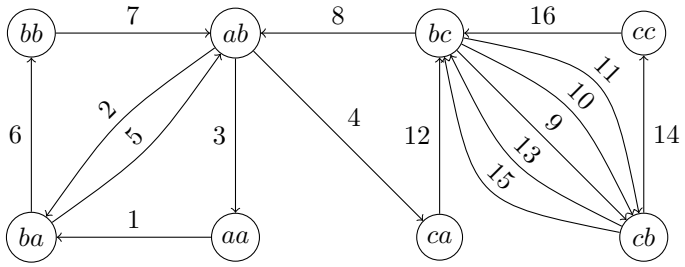
where $\mathcal{C}$ consists of all edges in $G_k(M_k([w]))$. □

Efficient implementations of the inverse transform rely on the fact that the k-order contexts of $M_k([w])$ are ordered. This allows the implementation of the k-order context graph G_k in a vectorized form [1,19,20,21].

Example 13. We compute the sort transform of order 2 of $w = bcbccbcbcabbaaba$ from Example 2. The list $M_2([w])$ is depicted in Figure 1(c). This yields the transform $(\mathrm{ST}_2(w), i) = (bbacabaacccbbcbb, 8)$ where $L = \mathrm{ST}_2(w)$ is the last column of the matrix $M_2([w])$ and w is the i-th element in $M_2([w])$. Next, we show how to reconstruct the input w from (L, i). The standard permutation induced by L is

$$\pi_L = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 3 & 5 & 7 & 8 & 1 & 2 & 6 & 12 & 13 & 15 & 16 & 4 & 9 & 10 & 11 & 14 \end{pmatrix}.$$

Note that π_L has four cycles $C_1 = (1, 3, 7, 6, 2, 5)$, $C_2 = (4, 8, 12)$, $C_3 = (9, 13)$, and $C_4 = (10, 15, 11, 16, 14)$. We obtain the context of order 2 of the j-th word by $c_j = \lambda_L \pi_L(j) \lambda_L \pi_L^2(j)$. In particular, $c_1 = aa$, $c_2 = c_3 = c_4 = ab$, $c_5 = c_6 = ba$, $c_7 = bb$, $c_8 = c_9 = c_{10} = c_{11} = bc$, $c_{12} = ca$, $c_{13} = c_{14} = c_{15} = cb$, and $c_{16} = cc$. With L and these contexts we can construct the graph $G = G_2(M_2([w]))$. The vertices of G are the contexts and the edge-labels represent positions in L. The graph G is depicted below:



We are starting at the context $c_i = c_8 = bc$ and then we are traversing G along the smallest edge-label amongst the unused edges. The sequence of the edge labels obtained this way is

$$(8, 2, 5, 3, 1, 6, 7, 4, 12, 9, 13, 10, 14, 16, 11, 15).$$

The labeling of this sequence of positions yields $\overline{w} = abaabbacbcbccbcb$. Since we are constructing the input from right to left, we obtain $w = bcbccbcbcabbaaba$.

6 The bijective sort transform

The bijective sort transform combines the Lyndon factorization with the ST. This yields a new algorithm which serves as a similar preprocessing step in data compression as the BWT. In a lot of applications, it can be used as a substitute for the ST. The proof of the bijectivity of the transform is slightly more technical than the analogous result for the bijective BWT. The main reason is that the bijective sort transform is less modular than the bijective BWT (which can be grouped into a 'Lyndon factorization part' and a 'Gessel-Reutenauer transform part' and which for example allows the use of different orders on the alphabet for the different parts).

For the description of the bijective ST and of its inverse, we rely on notions from Section 5. The bijective ST of a word w of length n is defined as follows. Let $w = v_s \cdots v_1$ with $v_s \geq \cdots \geq v_1$ be the Lyndon factorization of w. Let $M_k([v_1], \ldots, [v_s]) = (u_1, \ldots, u_n)$. Then the bijective ST of order k of w is $\mathrm{LST}_k(w) = \mathrm{last}(u_1) \cdots \mathrm{last}(u_n)$. That is, we are sorting the conjugacy classes of the Lyndon factors by k-order contexts and then take the sequence of the last letters. The letter L in LST_k is for *Lyndon*.

Theorem 14. *The bijective ST of order k is invertible, i.e., given $\mathrm{LST}_k(w)$ one can reconstruct the word w.*

Proof. Let $w = v_s \cdots v_1$ with $v_s \geq \cdots \geq v_1$ be the Lyndon factorization of w, let $c_i = \mathrm{context}_k(v_i)$, and let $L = \mathrm{LST}_k(w)$. By Lemma 11 we can rebuild the k-order context graph $G = G_k(M_k([v_1], \ldots, [v_s])) = (w_1, \ldots, w_n)$ from L. Let $\mathcal{C}_1$ consist of all edges in G. Then by Lemma 10 we see that

$$\begin{aligned}(\mathcal{C}_1, c_1) &\xrightarrow{v_1} (\mathcal{C}_2, c_1)\\ &\vdots\\ (\mathcal{C}_s, c_s) &\xrightarrow{v_s} (\mathcal{C}_{s+1}, c_s).\end{aligned}$$

We cannot use this directly for the reconstruction of w since we do not know the Lyndon factors v_i and the contexts c_i.

The word v_1 is the first element in the list $M_k([v_1], \ldots, [v_s])$ because v_1 is lexicographically minimal and it appears as the first element in the list $([v_1], \ldots, [v_s])$. Therefore, by Lemma 3 we obtain $c_1 = \mathrm{context}_k(v_1) = \lambda_L \pi_L(1) \cdots \lambda_L \pi_L^k(1)$.

The reconstruction procedure works from right to left. Suppose we have already reconstructed $w' v_j \cdots v_1$ for $j \geq 0$ with w' being a (possibly empty) suffix of v_{j+1}. Moreover, suppose we have used the correct contexts $c_1, \ldots, c_{j+1}$. Consider the con-

figuration $(\mathcal{C}', c')$ defined by

$$\begin{aligned}(\mathcal{C}_1, c_1) &\xrightarrow{v_1} (\mathcal{C}_2, c_1)\\ &\vdots\\ (\mathcal{C}_j, c_j) &\xrightarrow{v_j} (\mathcal{C}_{j+1}, c_j)\\ (\mathcal{C}_{j+1}, c_{j+1}) &\xrightarrow{w'} (\mathcal{C}', c')\end{aligned}$$

We assume that the following invariant holds: $\mathcal{C}_{j+1}$ contains no edges (c'', ℓ, c''') with $c'' < c_{j+1}$. We want to rebuild the next letter. We have to consider three cases. First, if $|w'| < |v_{j+1}|$ then

$$(\mathcal{C}', c') \xrightarrow{a} (\mathcal{C}'', c'')$$

yields the next letter a such that aw' is a suffix of v_{j+1}. Second, let $|w'| = |v_{j+1}|$ and suppose that there exists an edge $(c_{j+1}, \ell, c''') \in \mathcal{C}'$ starting at $c' = c_{j+1}$. Then there exists a word v' in $[v_{j+2}], \ldots, [v_s]$ such that $\mathrm{context}_k(v') = c_{j+1}$. If $\mathrm{context}_k(v_{j+2}) \neq c_{j+1}$ then from the invariant it follows that $\mathrm{context}_k(v_{j+2}) > c_{j+1} = \mathrm{context}_k(v')$. This is a contradiction, since v_{j+2} is minimal among the words in $[v_{j+2}], \ldots, [v_s]$. Hence, $\mathrm{context}_k(v_{j+2}) = c_{j+2} = c_{j+1}$ and the invariant still holds for $\mathcal{C}_{j+2} = \mathcal{C}'$. The last letter a of v_{j+2} is obtained by

$$(\mathcal{C}', c') = (\mathcal{C}_{j+2}, c_{j+2}) \xrightarrow{a} (\mathcal{C}'', c'').$$

The third case is $|w'| = |v_{j+1}|$ and there is no edge $(c_{j+1}, \ell, c''') \in \mathcal{C}'$ starting at $c' = c_{j+1}$. As before, v_{j+2} is minimal among the (remaining) words in $[v_{j+2}], \ldots, [v_s]$. By construction of G, the unique edge $(c'', \ell, c''') \in \mathcal{C}'$ with the minimal label ℓ has the property that $w_\ell = v_{j+2}$. In particular, $c'' = c_{j+2}$. Since v_{j+2} is minimal, the invariant for $\mathcal{C}_{j+2} = \mathcal{C}'$ is established. In this case, the last letter a of v_{j+2} is obtained by

$$(\mathcal{C}_{j+2}, c_{j+2}) \xrightarrow{a} (\mathcal{C}'', c''').$$

We note that we cannot distinguish between the first and the second case since we do not know the length of v_{j+1}, but in both cases, the computation of the next symbol is identical. In particular, in contrast to the bijective BWT we do not implicitly recover the Lyndon factorization of w. □

We note that the proof of Theorem 14 heavily relies on two design criteria. The first one is to consider $M_k([v_1], \ldots, [v_s])$ rather than $M_k([v_s], \ldots, [v_1])$, and the second is to use right-shifts rather than left-shifts. The proof of Theorem 14 yields the following algorithm for reconstructing w from $L = \mathrm{LST}_k(w)$:

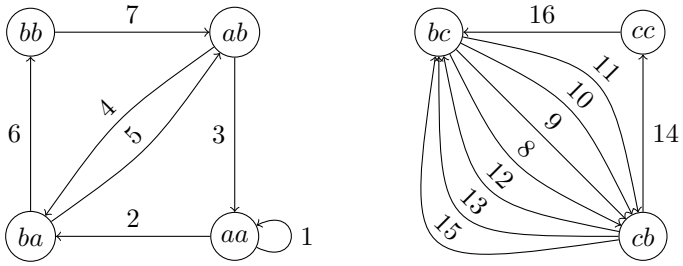
(1) Compute the k-order context graph $G = G_k$ and the k-order context c_1 of the last Lyndon factor of w.
(2) Start with the configuration $(\mathcal{C}, c)$ where $\mathcal{C}$ contains all edges of G and $c := c_1$.
(3) If there exists an outgoing edge starting at c in the set $\mathcal{C}$, then
   – Let (c, ℓ, c') be the edge with the minimal label ℓ starting at c.
   – Output $\lambda_L(\ell)$.
   – Set $\mathcal{C} := \mathcal{C} \setminus \{(c, \ell, c')\}$ and $c := c'$.
   – Continue with step (3).
(4) If there is no outgoing edge starting at c in the set $\mathcal{C}$, but $\mathcal{C} \neq \emptyset$, then
   – Let $(c', \ell, c'') \in \mathcal{C}$ be the edge with the minimal label ℓ.

- Output $\lambda_L(\ell)$.
- Set $\mathcal{C} := \mathcal{C} \setminus \{(c', \ell, c'')\}$ and $c := c''$.
- Continue with step (3).

(5) The algorithm terminates as soon as $\mathcal{C} = \emptyset$.

The sequence of the outputs is the reversal $\overline{w}$ of the word w.

Example 15. We consider the word $w = bcbccbcbcabbaaba$ from Example 2 and its Lyndon factorization $w = v_6 \cdots v_1$ where $v_6 = bcbcc$, $v_5 = bc$, $v_4 = bc$, $v_3 = abb$, $v_2 = aab$, and $v_1 = a$. For this particular word w the bijective Burrows-Wheeler transform and the bijective sort transform of order 2 coincide. From Example 9, we know $L = \mathrm{LST}_2(w) = \mathrm{BWTS}(w) = abababaccccbbcbb$ and the standard permutation π_L. As in Example 13 we can reconstruct the 2-order contexts $c_1, \ldots, c_{16}$ of $M_2([v_1], \ldots, [v_6])$: $c_1 = c_2 = aa$, $c_3 = c_4 = ab$, $c_5 = c_6 = ba$, $c_7 = bb$, $c_8 = c_9 = c_{10} = c_{11} = bc$, $c_{12} = c_{13} = c_{14} = c_{15} = cb$, and $c_{16} = cc$. With L and the 2-order contexts we can construct the graph $G = G_k(M_2([v_1], \ldots, [v_6]))$:



We are starting with the edge with label 1 and then we are traversing G along the smallest unused edges. If we end in a context with no outgoing unused edges, then we are continuing with the smallest unused edge. This gives the sequence $(1, 2, 5, 3)$ after which we end in context aa with no unused edges available. Then we continue with the sequences $(4, 6, 7)$ and $(8, 12, 9, 13, 10, 14, 16, 11, 15)$. The complete sequence of edge labels obtained this way is

$$(1, 2, 5, 3,\ 4, 6, 7,\ 8, 12, 9, 13, 10, 14, 16, 11, 15)$$

and the labeling of this sequence with λ_L yields $\overline{w} = abaabbacbcbccbcb$. As for the ST, we are reconstructing the input from right to left, and hence we get $w = bcbccbcbcabbaaba$.

7 Summary

We discussed two bijective variants of the Burrows-Wheeler transform (BWT). The first one is due to Scott. Roughly speaking, it is a combination of the Lyndon factorization and the Gessel-Reuternauer transform. The second variant is derived from the sort transform (ST); it is the main contribution of this paper. We gave full constructive proofs for the bijectivity of both transforms. As a by-product, we provided algorithms for the inverse of the BWT and the inverse of the ST. For the latter, we introduced an auxiliary graph structure — the k-order context graph. This graph yields an intermediate step in the computation of the inverse of the ST and the bijective ST. It can be seen as a generalization of the cycle decomposition of the standard permutation — which in turn can be used as an intermediate step in the computation of the inverse of the BWT and the bijective BWT.

Acknowledgments. The author would like to thank Yossi Gil and David A. Scott for many helpful discussions on this topic as well as Alexander Lauser, Antonio Restivo, and the anonymous referees for their numerous suggestions which improved the presentation of this paper.

References

1. D. Adjeroh, T. Bell, and A. Mukherjee: *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching*, Springer Publishing Company, Incorporated, 2008.
2. Z. Arnavut and M. Arnavut: *Investigation of block-sorting of multiset permutations.* Int. J. Comput. Math., 81(10) 2004, pp. 1213–1222.
3. B. Balkenhol and S. Kurtz: *Universal data compression based on the Burrows-Wheeler transformation: Theory and practice.* IEEE Trans. Computers, 49(10) 2000, pp. 1043–1053.
4. T. Bell, I. H. Witten, and J. G. Cleary: *Modeling for text compression.* ACM Comput. Surv., 21(4) 1989, pp. 557–591.
5. M. Burrows and D. J. Wheeler: *A block-sorting lossless data compression algorithm*, Tech. Rep. 124, Digital SRC Research Report, 1994.
6. K. T. Chen, R. H. Fox, and R. C. Lyndon: *Free differential calculus, IV — The quotient groups of the lower central series.* Ann. Math., 68(1) 1958, pp. 81–95.
7. J. G. Cleary and I. H. Witten: *Data compression using adaptive coding and partial string matching.* IEEE Trans. Commun., 32(4) 1984, pp. 396–402.
8. M. Crochemore, J. Désarménien, and D. Perrin: *A note on the Burrows-Wheeler transformation.* Theor. Comput. Sci., 332(1-3) 2005, pp. 567–572.
9. J.-P. Duval: *Factorizing words over an ordered alphabet.* J. Algorithms, 4(4) 1983, pp. 363–381.
10. N. J. Fine and H. S. Wilf: *Uniqueness theorems for periodic functions.* Proc. Amer. Math. Soc., 16 1965, pp. 109–114.
11. I. M. Gessel and C. Reutenauer: *Counting permutations with given cycle structure and descent set.* J. Comb. Theory, Ser. A, 64(2) 1993, pp. 189–215.
12. J. Gil and D. A. Scott: *A bijective string sorting transform*, submitted.
13. A. Lempel and J. Ziv: *A universal algorithm for sequential data compression.* IEEE Trans. Inform. Theory, 23(3) 1977, pp. 337–343.
14. A. Lempel and J. Ziv: *Compression of individual sequences via variable-rate coding.* IEEE Trans. Inform. Theory, 24(5) 1978, pp. 530–536.
15. M. Lothaire, ed., *Combinatorics on Words*, Addison-Wesley, Reading, MA, 1983.
16. S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino: *An extension of the Burrows Wheeler transform and applications to sequence comparison and data compression*, in Combinatorial Pattern Matching, CPM 2005, Proceedings, vol. 3537 of LNCS, Springer, 2005, pp. 178–189.
17. S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino: *An extension of the Burrows-Wheeler transform.* Theor. Comput. Sci., 387(3) 2007, pp. 298–312.
18. G. Manzini: *An analysis of the Burrows-Wheeler transform.* Journal of the ACM, 48(3) 2001, pp. 407–430.
19. G. Nong and S. Zhang: *Unifying the Burrows-Wheeler and the Schindler transforms*, in Data Compression Conference, DCC 2006. Proceedings, IEEE Computer Society, 2006, p. 464.
20. G. Nong and S. Zhang: *Efficient algorithms for the inverse sort transform.* IEEE Trans. Computers, 56(11) 2007, pp. 1564–1574.
21. G. Nong, S. Zhang, and W. H. Chan: *Computing inverse ST in linear complexity*, in Combinatorial Pattern Matching, CPM 2008, Proceedings, vol. 5029 of LNCS, Springer, 2008, pp. 178–190.
22. M. Schindler: *A fast block-sorting algorithm for lossless data compression*, in Data Compression Conference, DCC 1997. Proceedings, IEEE Computer Society, 1997, p. 469.
23. D. A. Scott: *Personal communication*, 2009.