Novel and Generalized Sort-based Transform for Lossless Data Compression

Abstract. We propose a new sort-based transform for lossless data compression that can replace the BWT transform in the block-sorting data compression algorithm. The proposed transform is a parametric generalization of the BWT and the RadixZip transform proposed by Vo and Manku, which is a rather new variation of the BWT. For a class of parameters, the transform can be run in time linear in the data length. We give an asymptotic compression bound attained by our algorithm.

1 Introduction

The block-sorting data compression algorithm [4] has been analyzed and evaluated both theoretically and empirically by researchers from the fields of information theory and algorithms. Several extensions to this algorithm and applications have been developed for various purposes [1]. Most of these extensions are modifications and generalizations of the *BWT* (the *Burrows–Wheeler Transform*), which is the core component of the block-sorting data compression algorithm. Few transformations that are completely different from the BWT have been developed. One such recent example is the *RadixZip Transform* proposed by Vo and Manku [9], which can replace the BWT in the block-sorting data compression algorithm.

In this paper, we propose a parametric generalization of the following two different transforms: the BWT and the *permute transform* in RadixZip. The proposed transform, called the *generalized radix permute* transform, or the *GRP* transform, bridges the two existing transforms. It also includes some of the finite-order variations of the BWT [8], [7] as special cases. While the original BWT uses unlimited order contexts, RadixZip uses only the contexts of orders from zero to a predetermined upperbound. Since RadixZip cannot exclude low-order contexts, it cannot obtain the high-order statistics of source strings.

In our GRP transform, the lowest order at which the encoder begins to obtain the statistics of source strings can be selected arbitrarily. The proposed transform is more generalized than the finite-order variations of the BWT since both the highest and lowest orders of contexts can be controlled. In the transform, the contexts from the shortest to the longest are cyclically adopted to predict the following symbols. Other remarkable characteristics are:

- As long as the lowest order remains constant, both the forward and inverse transformations run in time linear in the string length.
- By incorporating an appropriate second-step actual encoder, we can show that for sufficiently long strings from a stationary and ergodic source, the

average codeword length per source symbol attained by the proposed transform converges to the entropy rate of the source within at most one extra bit.

In this paper, we concentrate on presenting the GRP transform itself and its asymptotic analysis in compression performance. The GRP transform can be applied to any data of any length. However, we present its simplest version for simplicity, in which we require the data lengths to be integer multiples of a parameter.

2 GRP Transform

2.1 Preliminaries

Let

$$x[1:n] = x_1 x_2 \cdots x_n$$

be an n-symbol string over an ordered alphabet A of size $|A|$. For integers $i > j$, the string $x[i:j]$ denotes the empty string. The string $x[1:n]$ will be denoted also as x_1^n in the later analysis section. Similarly, a two-dimensional $n \times m$ matrix M of symbols is denoted by $M[1:n][1:m]$.

Similar to the BWT, the GRP transform converts the input string $x[1:n]$ to another string $y[1:n] \in A^n$ and an integer L. The GRP transform has two integer parameters. The first parameter is called the *block* length, which is denoted by ℓ. For simplicity it is assumed that the string length n is an integer multiple of ℓ, that is, $n = b\ell$ for an integer b.

In our transform, the input string is divided into b non-overlapping blocks of length ℓ, and saved as the column vectors of a matrix as follows:

$$T[1:\ell][1:b] = \begin{bmatrix} x_1 & x_{\ell+1} & x_{2\ell+1} & \cdots & x_{(b-1)\ell+1} \\ x_2 & x_{\ell+2} & x_{2\ell+2} & \cdots & x_{(b-1)\ell+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_\ell & x_{2\ell} & x_{3\ell} & \cdots & x_{b\ell} \end{bmatrix}. \tag{1}$$

The second parameter of the GRP transform is called the *context order*, or simply *order*, which is a non-negative integer less than or equal to ℓ. Let d denote the order. We first perform the left-cyclic shift of the top d rows of $T[1:\ell][1:b]$ and insert the results as the bottom rows of $T[1:\ell][1:b]$. Thus, the GRP transform is applied to the initial configuration of the $(\ell + d) \times b$ matrix given below:

$$T[1:\ell+d][1:b] = \begin{bmatrix} x_1 & x_{\ell+1} & \cdots & x_{(b-1)\ell+1} \\ x_2 & x_{\ell+2} & \cdots & x_{(b-1)\ell+2} \\ \vdots & \vdots & \vdots & \vdots \\ x_\ell & x_{2\ell} & \cdots & x_{b\ell} \\ x_{\ell+1} & x_{2\ell+1} & \cdots & x_1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{\ell+d} & x_{2\ell+d} & \cdots & x_d \end{bmatrix}. \tag{2}$$

As an example, consider the string

$$x[1:15] = \texttt{hotspotstopshot}, \tag{3}$$

and let $\ell = 3$ and $d = 2$. Then, $b = 5$ and

$$T[1:5][1:5] = \begin{bmatrix} \texttt{h} & \texttt{s} & \texttt{t} & \texttt{o} & \texttt{h} \\ \texttt{o} & \texttt{p} & \texttt{s} & \texttt{p} & \texttt{o} \\ \texttt{t} & \texttt{o} & \texttt{t} & \texttt{s} & \texttt{t} \\ \texttt{s} & \texttt{t} & \texttt{o} & \texttt{h} & \texttt{h} \\ \texttt{p} & \texttt{s} & \texttt{p} & \texttt{o} & \texttt{o} \end{bmatrix}. \tag{4}$$

2.2 Forward Transformation

The forward transformation of the GRP transform proceeds as follows:

1. /* Initialization */
   Convert the input string $x[1:n]$ into a matrix $T = T[1:\ell+d][1:b]$;
   Set $\boldsymbol{v}$:= the rightmost column vector of T;
   Set $L := b$;
2. **for** $i := 1$ **to** d **do**
   (a) Sort the column vectors of T in a stable manner according to the symbols of the ith row;
   /* The vector $\boldsymbol{v}$ may have moved to another column. */
   (b) Set $L :=$ the current column number of $\boldsymbol{v}$;
   end for
3. **for** $i := d+1$ **to** $d+\ell$ **do**
   (a) Output the ith row of T;
   (b) **if** $i = d+\ell$ **then** break;
   (c) Sort the column vectors of T in a stable manner according to the symbols of the ith row;
   end for
4. Concatenate the outputs of Step 3 (a) to form $y[1:n] = y_1 y_2 \cdots y_n$. The string $y[1:n]$ with the value of L is an output of the GRP transform.

For the string given in (3), the above procedure works as follows:

Step 2: $i = 1$

Perform a stable sort on the columns of T using the first row as the key to yield

$$T = \begin{bmatrix} \texttt{h} & \texttt{h} & \texttt{o} & \texttt{s} & \texttt{t} \\ \texttt{o} & \texttt{o} & \texttt{p} & \texttt{p} & \texttt{s} \\ \texttt{t} & \texttt{t} & \texttt{s} & \texttt{o} & \texttt{t} \\ \texttt{s} & \texttt{h} & \texttt{h} & \texttt{t} & \texttt{o} \\ \texttt{p} & \texttt{o} & \texttt{o} & \texttt{s} & \texttt{p} \end{bmatrix}.$$

Now, the column $\boldsymbol{v}$ has shifted to the second column. Thus, we have $L = 2$.

$i = 2$

Perform a stable sort on the columns in T by using the second row. This does not change the value of T. Now, $L = 2$ is stored.

Step 3: $i = 3$

The third row of T, `ttsot`, is outputted. Then, perform a stable sort on the columns in T by using the third row to yield

$$T = \begin{bmatrix} \texttt{s} & \texttt{o} & \texttt{h} & \texttt{h} & \texttt{t} \\ \texttt{p} & \texttt{p} & \texttt{o} & \texttt{o} & \texttt{s} \\ \texttt{o} & \texttt{s} & \texttt{t} & \texttt{t} & \texttt{t} \\ \texttt{t} & \texttt{h} & \texttt{s} & \texttt{h} & \texttt{o} \\ \texttt{s} & \texttt{o} & \texttt{p} & \texttt{o} & \texttt{p} \end{bmatrix}.$$

$i = 4$

The fourth row of T, `thsho`, is outputted. Then, perform a stable sort on the columns in T by using the fourth row to yield

$$T = \begin{bmatrix} \texttt{o} & \texttt{h} & \texttt{t} & \texttt{h} & \texttt{s} \\ \texttt{p} & \texttt{o} & \texttt{s} & \texttt{o} & \texttt{p} \\ \texttt{s} & \texttt{t} & \texttt{t} & \texttt{t} & \texttt{o} \\ \texttt{h} & \texttt{h} & \texttt{o} & \texttt{s} & \texttt{t} \\ \texttt{o} & \texttt{o} & \texttt{p} & \texttt{p} & \texttt{s} \end{bmatrix}.$$

$i = 5$

The fifth row of T, `oopps`, is outputted. Since $i = \ell + d$ $(= 5)$, the concatenation of the above three outputs and the value of L yield

$$\begin{array}{l} y[1:15] = \texttt{ttsotthshoooppps}, \\ L = 2. \end{array} \tag{5}$$

This is the result of the GRP transform of the string given in (3).

2.3 Inverse Transformation

The GRP transform is reversible. The inverse transformation of the GRP transform is more complicated than the forward transformation. Actually, in its description below, we will introduce a couple of auxiliary matrices that have not appeared in the forward transformation. However, these matrices are used only for explaining the transformation and are not essential for the transformation. The values of the parameters ℓ and d, and n are the same in both the forward and inverse transformations. Hence, the number of blocks of the string, $b = n/\ell$, is an integer.

1. /* Initialization */
   Store the string $y[1:n]$ in an $\ell \times b$ matrix $S = S[1:\ell][1:b]$ according to

   $$S[i][j] := y[(i-1)b + j] \quad \text{for } 1 \le i \le \ell, 1 \le j \le b;$$

   Stack its ℓth row as the bottom row of an $(\ell + d) \times b$ matrix U;

2. **for** $j := 1$ **to** $\ell - 1$ **do**
   (a) Sort the symbols in the $(\ell - j)$th row of S alphabetically, and put the result into the $(\ell + d - j)$th row of U;
   (b) Sort the columns of U so that its $(\ell + d - j)$th row corresponds to the $(\ell - j)$th row of S;
   end for
3. (a) Copy the bottom d rows of U into a $d \times b$ matrix V;
   (b) Considering the bottom row of V to be a significant part of the key, perform a radix sort on the columns of V (that is, perform a stable sort on the columns of V using the first to dth rows as the keys in this order);
   (c) Stack the matrix V on U;
   /* Note that U is now identical to T which is obtained immediately after Step 2 in the forward transformation. */
4. Let $\boldsymbol{w}$ be the Lth column of U;
   Copy $\boldsymbol{w}$ to the bth column of an $(\ell + d) \times b$ matrix T;
5. **for** $j := 1$ **to** $b - 1$ **do**
   (a) From the columns of U that have not been copied to T, select the leftmost column that has the same d top symbols as the bottom d-symbol column of $\boldsymbol{w}$;
   (b) Set $\boldsymbol{w} :=$ the selected column, and copy it to T as the jth column;
   end for
6. /* The matrix T in (2) has been reconstructed. */
   Recover the original string by

$$x[i + (j-1)\ell] := T[i][j] \quad \text{for } 1 \le i \le \ell,\ 1 \le j \le b.$$

Before giving the general explanation of the reversibility of the above inverse transformation, we show how it works for the example given in (5).

Step 1

$$S = \begin{bmatrix} \mathtt{t} & \mathtt{t} & \mathtt{s} & \mathtt{o} & \mathtt{t} \\ \mathtt{t} & \mathtt{h} & \mathtt{s} & \mathtt{h} & \mathtt{o} \\ \mathtt{o} & \mathtt{o} & \mathtt{p} & \mathtt{p} & \mathtt{s} \end{bmatrix}, \quad U = \begin{matrix} \\ \\ \\ \\ 5\text{th} \end{matrix} \begin{bmatrix} \\ \\ \\ \vdots \\ \mathtt{o} & \mathtt{o} & \mathtt{p} & \mathtt{p} & \mathtt{s} \end{bmatrix}.$$

Step 2: $j = 1$

$$U = \begin{bmatrix} \\ \\ \\ \vdots \\ \mathtt{h} & \mathtt{h} & \mathtt{o} & \mathtt{s} & \mathtt{t} \\ \mathtt{o} & \mathtt{o} & \mathtt{p} & \mathtt{p} & \mathtt{s} \end{bmatrix} \longrightarrow U = \begin{bmatrix} \\ \\ \\ \vdots \\ \mathtt{t} & \mathtt{h} & \mathtt{s} & \mathtt{h} & \mathtt{o} \\ \mathtt{s} & \mathtt{o} & \mathtt{p} & \mathtt{o} & \mathtt{p} \end{bmatrix}.$$

$j = 2$

$$U = \begin{bmatrix} & \vdots & & & \\ \mathtt{o} & \mathtt{s} & \mathtt{t} & \mathtt{t} & \mathtt{t} \\ \mathtt{t} & \mathtt{h} & \mathtt{s} & \mathtt{h} & \mathtt{o} \\ \mathtt{s} & \mathtt{o} & \mathtt{p} & \mathtt{o} & \mathtt{p} \end{bmatrix} \longrightarrow U = \begin{bmatrix} & \vdots & & & \\ \mathtt{t} & \mathtt{t} & \mathtt{s} & \mathtt{o} & \mathtt{t} \\ \mathtt{s} & \mathtt{h} & \mathtt{h} & \mathtt{t} & \mathtt{o} \\ \mathtt{p} & \mathtt{o} & \mathtt{o} & \mathtt{s} & \mathtt{p} \end{bmatrix}.$$

Step 3

$$V = \begin{bmatrix} \mathtt{s} & \mathtt{h} & \mathtt{h} & \mathtt{t} & \mathtt{o} \\ \mathtt{p} & \mathtt{o} & \mathtt{o} & \mathtt{s} & \mathtt{p} \end{bmatrix} \longrightarrow V = \begin{bmatrix} \mathtt{h} & \mathtt{h} & \mathtt{o} & \mathtt{s} & \mathtt{t} \\ \mathtt{o} & \mathtt{o} & \mathtt{p} & \mathtt{p} & \mathtt{s} \end{bmatrix},$$

$$U = \begin{bmatrix} \mathtt{h} & \mathtt{h} & \mathtt{o} & \mathtt{s} & \mathtt{t} \\ \mathtt{o} & \mathtt{o} & \mathtt{p} & \mathtt{p} & \mathtt{s} \\ \mathtt{t} & \mathtt{t} & \mathtt{s} & \mathtt{o} & \mathtt{t} \\ \mathtt{s} & \mathtt{h} & \mathtt{h} & \mathtt{t} & \mathtt{o} \\ \mathtt{p} & \mathtt{o} & \mathtt{o} & \mathtt{s} & \mathtt{p} \end{bmatrix}.$$

Step 4

$$T = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \mathtt{h} \\ \cdot & \cdot & \cdot & \cdot & \mathtt{o} \\ \cdot & \cdot & \cdot & \cdot & \mathtt{t} \\ \cdot & \cdot & \cdot & \cdot & \mathtt{h} \\ \cdot & \cdot & \cdot & \cdot & \mathtt{o} \end{bmatrix}.$$

Step 5: $j = 1$ $j = 2$ $j = 3$

$$T = \begin{bmatrix} \mathtt{h} & \cdot & \cdot & \cdot & \mathtt{h} \\ \mathtt{o} & \cdot & \cdot & \cdot & \mathtt{o} \\ \mathtt{t} & \cdot & \cdot & \cdot & \mathtt{t} \\ \mathtt{s} & \cdot & \cdot & \cdot & \mathtt{h} \\ \mathtt{p} & \cdot & \cdot & \cdot & \mathtt{o} \end{bmatrix}, \qquad T = \begin{bmatrix} \mathtt{h} & \mathtt{s} & \cdot & \cdot & \mathtt{h} \\ \mathtt{o} & \mathtt{p} & \cdot & \cdot & \mathtt{o} \\ \mathtt{t} & \mathtt{o} & \cdot & \cdot & \mathtt{t} \\ \mathtt{s} & \mathtt{t} & \cdot & \cdot & \mathtt{h} \\ \mathtt{p} & \mathtt{s} & \cdot & \cdot & \mathtt{o} \end{bmatrix}. \qquad T = \begin{bmatrix} \mathtt{h} & \mathtt{s} & \mathtt{t} & \cdot & \mathtt{h} \\ \mathtt{o} & \mathtt{p} & \mathtt{s} & \cdot & \mathtt{o} \\ \mathtt{t} & \mathtt{o} & \mathtt{t} & \cdot & \mathtt{t} \\ \mathtt{s} & \mathtt{t} & \mathtt{o} & \cdot & \mathtt{h} \\ \mathtt{p} & \mathtt{s} & \mathtt{p} & \cdot & \mathtt{o} \end{bmatrix},$$

$j = 4$

$$T = \begin{bmatrix} \mathtt{h} & \mathtt{s} & \mathtt{t} & \mathtt{o} & \mathtt{h} \\ \mathtt{o} & \mathtt{p} & \mathtt{s} & \mathtt{p} & \mathtt{o} \\ \mathtt{t} & \mathtt{o} & \mathtt{t} & \mathtt{s} & \mathtt{t} \\ \mathtt{s} & \mathtt{t} & \mathtt{o} & \mathtt{h} & \mathtt{h} \\ \mathtt{p} & \mathtt{s} & \mathtt{p} & \mathtt{o} & \mathtt{o} \end{bmatrix}.$$

Step 6

$$x[1:15] = \mathtt{hotspotstopshot}.$$

2.4 Reversibility and Complexity

In order to show the reversibility of the GRP transform, we first note the symmetric relation between Step 3 of the forward transformation and Step 2 of the inverse transformation, which can be stated in the following lemma.

Lemma 1. *For i and j such that $i + j = d + \ell$, at the end of the jth iteration of the loop in Step 2 of the inverse transformation, the bottom $j + 1$ rows of U are identical to the bottom $d + \ell - i + 1$ rows of T in Step 3 (a) of the forward transformation.*

The above lemma can be proved by induction on j. The case of $j = 0$ corresponds to the initial state of the loop in Step 2 of the inverse transformation. In this state, the bottom row of U is simply a copy of the last output of Step 3 of the forward transformation. From the condition of the lemma, we have $i = d+\ell$ when $j = 0$, which corresponds to the last iteration of Step 3 of the forward transformation. Therefore, the statement of the lemma holds for $j = 0$. Starting from this initial state, we can show the validity of the statement from $j = 1$ to $j = \ell - 1$, inductively. Finally, we can show that, at the end of Step 2 of the inverse transformation, the bottom ℓ rows of U are identical to the bottom ℓ rows of T that are obtained immediately after Step 2 of the forward transformation.

In the inverse transformation, the process then moves on to Step 3, which is essentially the same as Step 2 of the forward transformation. Thus, we can establish the fact written as the comment in Step 3 of the inverse transformation that U and T are identical. The rest of the inverse transformation, namely Steps 4 and 5, can be easily validated by the stability of the sorting process of Step 2 of the forward transformation. In this way, we can prove the reversibility of the GRP transform.

Here, we make a brief comment about the time complexity of the GRP transform. We assume that each stable sorting process can be performed linearly by using bucket sorting. Under this assumption, the forward transformation can be done in $O(b(\ell + d)) = O(n + bd)$ time.

The inverse transformation seems more time-demanding than the forward transformation since Step 5 of the inverse transformation requires string searching. Actually, however, we can perform this process of string searching in $O(bd)$ time by using the result of Step 3 (b). In Step 5, for every column, say $\boldsymbol{w}$, of U, we must find a column that has the same d top symbols as the d-symbol bottom column of $\boldsymbol{w}$. Step 3 has already established the correspondence between every $\boldsymbol{w}$ and at least one such column. Moreover, after the step, all columns are arranged in lexicographic order of the top d-symbols. Therefore, it is not so difficult to find the column that satisfies the condition of Step 5. The total time required in Step 5 is proportional to the total number of symbols in the top d rows in T. In summary, we can prove the following theorem.

Theorem 1. *For any string of length n, both the forward and inverse transformations run in $O(n+bd)$ time, where b is the number of blocks of the string, and d is the context order of the GRP transform. For any fixed order d, therefore, they run in time linear in the string length n.*

Remark: In this paper, we have presented only the case of $n = b\ell$. We have already succeeded in eliminating this assumption. The GRP transform can be modified to be applicable to any string of any length. We have also assumed that the order d satisfies $0 \leq d \leq \ell$. The transform can be extended for larger values of d than ℓ so that it includes existing transforms as special cases. Specific correspondences follow.

GRP with $\ell = 1$ and $d = n$: BWT;
GRP with $\ell = 1$ and $d < n$: ST transform [8],[7];
GRP with $d = 0$: Permute transform in RadixZip.

3 Information Theoretical Analysis

3.1 Second-step Algorithm

Similar to the BWT, the GRP transform requires a second-step algorithm for actual compression. In addition to the same algorithms as those adopted in the block sorting compression algorithm [1], [5], we may incorporate new encoding methods that rely on the nature of the GRP transform. For example, the output string of the GRP transform is a concatenation of ℓ blocks; each block can be encoded by distinct encoding methods. In this paper, however, we consider only the simplest case for the analysis of the asymptotic performance of the proposed transform.

We encode the output $y[1:n]$ of the GRP transform by using the Move-to-Front (MTF) encoding scheme [3], which produces a list of integers from 1 to the size $|A|$ of the source alphabet. Then, we encode each integer in the list using the δ code of Elias [6]. The codeword length for integer t is upperbounded by

$$f(t) = \log t + 2\log(\log t + 1) + 1 \quad \text{bit.} \tag{6}$$

We will ignore the codeword for the integer component L of the output, for simplicity.

3.2 Asymptotic Characterization

Although the order d can be extended to an arbitrary integer as mentioned above, we restrict its range to $0 \leq d \leq \ell$. We first shift the blocks of the input string by d symbols. That is, we assume $x[(j-1)\ell + d + 1 : j\ell + d]$ to be the jth block $(1 \leq j \leq b-1)$. Thus, we consider only the substring $x[d+1:(b-1)\ell + d]$. We ignore $x[1:d]$, which serves only as the context to the following symbols in the first column of matrix T in (2), and is encoded in the last column in a virtual context. We focus on the kth symbol $x[(j-1)\ell + d + k]$ in the jth block $(1 \leq k \leq \ell)$. We define the context of this kth symbol by $x[(j-1)\ell + 1 : (j-1)\ell + d + k - 1]$. The context of the kth symbol in the jth block is a substring of $d+k-1$ symbols that immediately precedes $x[(j-1)\ell + d + k]$. In the forward transformation, each kth symbol appears in the $(d+k)$th row of T, and is included somewhere

in $y[(k-1)b+1:kb]$ of the transformed string. Note that when the kth symbols $\{x[(j-1)\ell+d+k]\}_{j=1}^{b-1}$ are transformed into $y[(k-1)b+1:kb]$ in Step 3 of the forward transformation, their contexts are lexicographically arranged as columns consisting of top $d+k-1$ rows of T. That is, the same contexts appear consecutively as columns in T (see Fig. 1).

In Fig. 1, $y_{1|c}$ is the ith symbol of $y[(k-1)b+1:kb]$ that appeared in context c. Thus, $y_{1|c}$, $y_{2|c}$, $y_{3|c}, \ldots, y_{N(c)|c}$ are the symbols that appear sequentially in this order in context c in the transformed string, where $N(c)$ is the number of blocks that have the same prefix c. In general, for an arbitrary string $a_1^i = a_1 a_2 \cdots a_i \in A^i$ $(0 \le i \le \ell)$, $N(a_1^i)$ represents the number of blocks appeared in the entire $b-1$ blocks that begin with the prefix a_1^i. For the empty string λ, $N(\lambda)$ equals $b-1$. Let $z_1^{N(c)} = z_1 z_2 \cdots z_{N(c)}$ be a sequence of positive integers that is obtained from $y_{1|c} y_{2|c} \cdots y_{N(c)|c}$ by using the MTF scheme. For every symbol $a \in A$, if $y_{i|c}$ equals a with $i = t_1, t_2, \ldots, t_{N(ca)}$, then we have

$$\begin{aligned} z_{t_1} &\le |A|, \\ z_{t_i} &\le t_i - t_{i-1} \quad \text{for } 2 \le i \le N(ca). \end{aligned}$$

According to [3] and [2], the sum of the lengths of the codewords representing the symbol a in context c can be bounded by

$$f(|A|) + \sum_{i=2}^{N(ca)} f(t_i - t_{i-1}) \le N(ca)\, f\Big(\frac{N(c)+|A|}{N(ca)}\Big). \tag{7}$$

The kth symbols $\{x[(j-1)\ell+d+k]\}_{j=1}^{b-1}$ are transformed into $b-1$ symbols in $y[(k-1)b+1:kb]$, and then converted into a sequence of integers by the MTF scheme. Let $l_k(y_{(k)}^{b-1})$ denote the sum of the codeword lengths representing the $b-1$ kth symbols. Then, we have the following result, which is a direct consequence of the inequality (7).

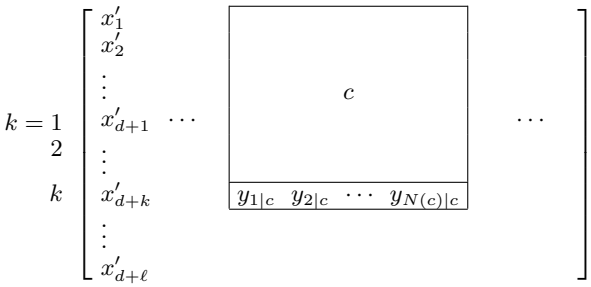

Fig. 1. Matrix T after transformation of kth symbols of blocks in Step 3 of the forward transformation.

Lemma 2. *For any fixed integer k in $[1, \ell]$, the kth symbols $\{x[(j-1)\ell + d + k]\}_{j=1}^{b-1}$ of $b-1$ blocks can be encoded with the length $l_k(y_{(k)}^{b-1})$, which satisfies*

$$\begin{aligned} l_k(y_{(k)}^{b-1}) &\le \sum_{a_1^{d+k-1}} \sum_{a_{d+k}} N(a_1^{d+k}) f\Big(\frac{N(a_1^{d+k-1}) + |A|}{N(a_1^{d+k})}\Big) \\ &= \sum_{a_1^i \in A^i} N(a_1^i)\, f\Big(\frac{N(a_1^{i-1}) + |A|}{N(a_1^i)}\Big) \quad \textit{for} \quad i = d+k, \end{aligned} \tag{8}$$

where the second summation is taken over a_{d+k} so that $N(a_1^{d+k})$ is greater than zero.

Suppose that an input string is generated from a stationary and ergodic source $\{X_i\}_{i=1}^{\infty}$ with the probability measure p and entropy rate H, where X_i takes values in the alphabet A. Let $p(a_1^m)$ denote the probability that X_1^m is equal to $a_1^m \in A^m$, and $p(a_m \mid a_1^{m-1})$ denote the conditional probability of a_m given a_1^{m-1}. Similarly, $p(a_{d+1}^{d+\ell} \mid a_1^d)$ represents

$$p(a_{d+1}^{d+\ell} \mid a_1^d) = \prod_{i=1}^{\ell} p(a_{d+i} \mid a_1^{d+i-1}) = \frac{p(a_1^{d+\ell})}{p(a_1^d)}. \tag{9}$$

The conditional joint entropy $H(X_{d+1}^{d+\ell} \mid X_1^d)$ is defined by

$$H(X_{d+1}^{d+\ell} \mid X_1^d) = -\sum_{a_1^d \in A^d} p(a_1^d) \sum_{p(a_{d+1}^{d+\ell} \mid a_1^d) \neq 0} p(a_{d+1}^{d+\ell} \mid a_1^d) \log p(a_{d+1}^{d+\ell} \mid a_1^d).$$

Then, we have

$$H = \lim_{d \to \infty} \frac{1}{\ell} H(X_{d+1}^{d+\ell} \mid X_1^d) \quad \text{for any } \ell \tag{10}$$

$$= \lim_{\ell \to \infty} \frac{1}{\ell} H(X_{d+1}^{d+\ell} \mid X_1^d) \quad \text{for any } d. \tag{11}$$

For the arbitrary fixed integers $\ell > 0$ and $b > 1$, consider a prefix of length $(b-1)\ell$ that begins at the $(d+1)$th place of an infinite string x over A. Divide the prefix into $b-1$ blocks of non-overlapping substrings each of length ℓ, and let $N_x(a_1^i)$ represent the number of blocks whose prefix is equal to a_1^i. Define a set

$$\tilde{D}_b(a_1^i, \varepsilon) = \left\{ x \in A^{\infty} : \left| \frac{N_x(a_1^i)}{b-1} - p(a_1^i) \right| > \varepsilon p(a_1^i) \right\} \tag{12}$$

for fixed b and $\varepsilon > 0$. Moreover, we introduce the following set:

$$D_b(d, \ell, \varepsilon) = \Big\{ \bigcup_{a_1^d} \tilde{D}_b(a_1^d, \varepsilon) \Big\} \cup \Big\{ \bigcup_{a_1^{d+\ell}} \tilde{D}_b(a_1^{d+\ell}, \varepsilon) \Big\}. \tag{13}$$

When we encode a $b\ell$-symbol prefix of x by using the proposed scheme, we represent the codeword length corresponding to the substring $x[d+1:(b-1)\ell+d]$ by $l(y^{b-1})$. That is,

$$l(y^{b-1}) = \sum_{k=1}^{\ell} l_k(y_{(k)}^{b-1}). \tag{14}$$

We can now bound the codeword length for each source symbol in our encoding scheme in the following way.

Theorem 2. *For arbitrarily fixed $\ell > 0$, $d \leq \ell$, and $\varepsilon > 0$, and for $x \notin D_b(d,\ell,\varepsilon)$, there exists a positive integer $B = B(d,\ell,\varepsilon)$ such that for any $b > B$,*

$$\frac{l(y^{b-1})}{(b-1)\ell} \;\leq\; \frac{1}{\ell} H(X_{d+1}^{d+\ell} \mid X_1^d) + \frac{2}{\ell} \log(H(X_{d+1}^{d+\ell} \mid X_1^d) + 1) + 1 + \hat{\varepsilon}, \tag{15}$$

where $\hat{\varepsilon} \to 0$ as $\varepsilon \to 0$.

Outline of Proof: From equations (6), (8), and (14), we have

$$\begin{aligned} l(y^{b-1}) &= \sum_{k=1}^{\ell} l_k(y_{(k)}^{b-1}) = \sum_{i=d+1}^{d+\ell} \sum_{a_1^i \in A^i} N(a_1^i)\, f\Big(\frac{N(a_1^{i-1}) + |A|}{N(a_1^i)}\Big) \\ &= \sum_{i=d+1}^{d+\ell} \sum_{a_1^i \in A^i} N(a_1^i) \left(\log \frac{N(a_1^{i-1}) + |A|}{N(a_1^i)} + 2 \log \left(\log \frac{N(a_1^{i-1}) + |A|}{N(a_1^i)} + 1 \right) + 1 \right). \end{aligned} \tag{16}$$

From the concavity of the logarithmic function, Jensen's inequality, and the boundedness on the number of prefixes of any length, we can give an upper bound on the first term in the last sum. The bound can be used to show that for any a_1^d, a_{d+1}^{ℓ}, and $x \notin D_b(d,\ell,\varepsilon)$,

$$\begin{aligned} &\frac{1}{b-1} \sum_{a_1^{d+\ell} \in A^{d+\ell}} N_x(a_1^{d+\ell}) \log \frac{N_x(a_1^d) + |A|}{N_x(a_1^{d+\ell})} \\ &\leq \sum_{a_1^d} p(a_1^d) \sum_{a_{d+1}^{d+\ell}} p(a_{d+1}^{d+\ell} \mid a_1^d) \log \frac{1}{p(a_{d+1}^{d+\ell} \mid a_1^d)} + \varepsilon' \\ &= H(X_{d+1}^{d+\ell} \mid X_1^d) + \varepsilon' \end{aligned} \tag{17}$$

for some ε' such that $\varepsilon' \to 0$ as $\varepsilon \to 0$. Applying a similar technique to the second logarithmic term in (16), we can show that for any a_1^d, a_{d+1}^{ℓ}, and $x \notin D_b(d,\ell,\varepsilon)$,

$$\begin{aligned} &\frac{2}{b-1} \sum_{i=d+1}^{d+\ell} \sum_{a_1^i \in A^i} N_x(a_1^i) \log \Big(\log \frac{N_x(a_1^{i-1}) + |A|}{N_x(a_1^i)} + 1 \Big) \\ &\qquad \leq 2 \log(H(X_{d+1}^{d+\ell} \mid X_1^d) + 1) + \varepsilon'' \end{aligned} \tag{18}$$

for some ε'' such that $\varepsilon'' \to 0$ as $\varepsilon \to 0$. By combining the inequalities (16), (17), and (18), we prove the theorem.

Lemma 3. *For any stationary and ergodic source p,*

$$p\Big(\bigcap_{\beta=1}^{\infty}\bigcup_{b=\beta}^{\infty}\tilde{D}_b(d,\ell,\varepsilon)\Big)=0 \text{ for any } d,\ \ell, \text{ and } \varepsilon. \tag{19}$$

Combining equation (11), Theorem 2, and Lemma 3, we have the following theorem.

Theorem 3. *For any stationary and ergodic source with entropy rate H, the codeword length per symbol satisfies*

$$\lim_{\substack{b\to\infty\\ \ell\to\infty}}\frac{l(y^{b-1})}{(b-1)\ell}\le H+1 \tag{20}$$

with probability one.

4 Conclusion

We have proposed a sort-based transform, called the GRP transform, which is a parametric generalization of the BWT. Although the average codeword length per source symbol attained by the proposed transform can be shown to converge to the entropy rate of the source within at most one extra bit for, e.g., $b=O(\sqrt{n})$ and $\ell=O(\sqrt{n})$, as $n\to\infty$, these parameters have to be optimized from a practical viewpoint. We are conducting empirical studies to find out the optimal values of the parameters and a better second-step algorithm.

References

1. Adjeroh, D., Bell, T., Mukherjee, A.: The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching, Springer (2008)
2. Arimura, M., Yamamoto, H.: Asymptotic optimality of the block sorting data compression algorithm, IEICE Trans. Fundamentals, E81-A (10), 2117–2122 (1998)
3. Bentley, J. L., Sleator, D. D., Tarjan, R. E., Wei V. K.: A locally adaptive data compression scheme, Comm. ACM, 29 (4), 320–330 (1986)
4. Burrows, M., Wheeler, D. J.: A block-sorting lossless data compression algorithm, SRC Research Report, 124 (1994)
5. Deorowicz, S.: Improvements to Burrows–Wheeler compression algorithm, Software—Practice and Experience, 30 (13), 1465–1483 (2000)
6. Elias, P.: Universal codeword sets and representations of the integers, IEEE Trans. Inform. Theory, IT-21, 194-203 (1975)
7. Nong, G., Zhang, S., Chan, W. H.: Computing inverse ST in linear complexity, Combinatorial Pattern Matching: 19th Annual Sympo., CPM2008, Pisa, 178–190 (2008)
8. Schindler, M.: A fast block-sorting algorithm for lossless data compression, DCC '97, Proc. Data Compression Conf., 469, Snowbird, UT. (1979)
9. Vo B. D., Manku, G. S.: RadixZip: Linear time compression of token streams, Very Large Data Bases: Proc. 33rd Intern. Conf. on Very Large Data Bases, Vienna, 1162–1172 (2007)