A PROOF AND IMPROVEMENT ON BURROWS-WHEELER TRANSFORM

M. R. Kabiri, S. Moavenat, H. Amindavar

Amirkabir University of Technology,
Department of Electrical Engineering, 424 Hafez, Tehran , Iran
mrkabiri@yahoo.com, moavenat@yahoo.com,
hamidami@cic.aut.ac.ir

ABSTRACT

One of the lossless compression methods make use of a transform proposed by Burrows and Wheeler (BWT)[1]. This transform is based on sorting of all cyclic shifts of the input sequence. In this paper we prove that the original sorting by BWT is unique for one large class of sorting methods that consider all characters in the input sequence for sorting. Our proof is based on a general algorithm for generating a contrary example for the other sorting methods in this class. Also we improve the compression ratio utilizing BWT by presenting a new method.

1. INTRODUCTION

BWT transformation proposes a simple method for lossless data compression. It assumes that the preceding characters of similar sequences are the same with great probability. It generates all the cyclic shifts of the input sequence, then it sorts these shifts lexicographically. By so doing, two cyclic shifts come together when some of their first characters are similar. By noticing that, if two substrings have same characters at the beginning, then the probability of their preceding characters to be the same will be high, the sequence consisting of the last characters has the property that the similar characters will be grouped together. After these steps, the methods such as Move To Front coding (MTF)[1] can be used to prepare the data for an entropy coder such as Huffman or Arithmetic coders [1]. Eventually, this causes better compression ratios than the traditional methods such as Lempel-Ziv [7]. One way to improve compression ratio is to consider the second step of the BWT[2]. It is also possible to improve the compression ratio by modifing the sort that BWT is based on (see [3], [6]). Although Burrows and Wheeler present a method for finding input sequence from the output sequence, there has not been a proof of why in BWT, no two input sequences will map onto the same output. In this paper we prove, two different sequences will not map onto the same output under BWT transform. Also there has been no successful attempt to change the order of sorting method used in BWT. In this paper we prove that BWT sorting method is the only reversible method that can be used among the class of sorting methods that make use of all characters in a sequence. Also we provide a new method in sorting order of BWT that makes use of an undiscovered notion in the first characters of the cyclic shifts of the input sequence in BWT matrix. This notion is the existence of a discontinuity in the sorting order of BWT. We show that this new method increases the rate of compression in text files, and we suggest a way to optimize it. This paper consists of the following sections. In section 2 we prove the uniqueness of BWT sorting method, in section 3 we propose a method to improve compression ratio utilizing BWT and at the end we provide conclusions.

2. UNIQUENESS OF BWT SORTING METHOD

Now, we discuss why changing the sorting method may improve the compression ratio. Also we prove that the only sorting strategy that can be used at the first step is the one used in [1]. Subsequently we prove that the other methods which make use of all characters for sorting, are not reversible. We first show that there are no two sequences that will map onto the same output under the BWT transform. And also we provide a general approach for obtaining two sequences that will map onto a same output, for any other sorting method rather than the one used in BWT. In [6] a sorting method that uses some characters of a stream and still is reversible is considered.

BWT transform puts each cyclic shift as a row of a matrix and sorts these rows lexicographically. This transform is based on the notion that if we have two similar substrings in a string, the probability that the preceding characters of these substrings are the same is high. However, it is probable that there exist two similar substrings that their preceding characters are not the same. For increasing the probability of similarity, we bring together characters whose preceding and consequent substrings are the same. In fact, for sorting, we make use of the first character and then one to the last character, and therefore, we consider one character from the first of the sequence of each cycle and one from the last, and thereof. We call this method MKT. As an example of MKT MKT($mississippi$)=$pssmpissiii$ (see Figure 1). If we do this, we come to the results that is shown in Table 1 (in this Table the column BWT94 is the results of BWT in bit/character that mentioned in [1] and MKT is the above method).

We see that the compression ratio is increased considerably. The example presented in Figure 1 is one reason to provide an incentive to consider changing the sort order of BWT for an improvement. However, this method of sorting is not reversible. In this section we prove that the sorting method used in BWT is the only reversible sorting that is possible. For this we prove the following theorems. We prove that the sorting method used in BWT is reversible and then we prove that for every other sorting method in the specified class, there exist at least two input sequences that will map onto the same output, thus, this sorting method is not reversible. In providing the following theorems we define: $S_1 \overset{rot}{=} S_2$ if and only if S_1 is equal to S_2 or S_1

sort order $\dashrightarrow$		$13 \;\cdots\; 42$
mississippi•		i•mississipp
•mississippi		ippi•mississ
i•mississipp		issippi•miss
pi•mississip		ississippi•m
ppi•mississi		mississippi•
ippi•mississ	$\overset{MKT}{\Longrightarrow}$	pi•mississip
sippi•missis		ppi•mississi
ssippi•missi		sissippi•mis
issippi•miss		sippi•missis
sissippi•mis		ssissippi•mi
ssissippi•mi		ssippi•missi
ississippi•m		•mississippi

Figure 1: Example of MKT algorithm for 'mississipi'

is a cyclic shift of S_2.

2.1. Theorem 1

If S_1 and S_2 are two strings that will map onto the same output under BWT, then $S_1 \overset{rot}{=} S_2$.

2.1.1. Proof :

Suppose $S_1 = a_n a_{n-1} \ldots a_1$ and $S_2 = b_n b_{n-1} \ldots b_1$. We construct the following matrices:

$$M_1 = \begin{pmatrix} a_n & a_{n-1} & . & . & . & a_1 \\ a_1 & a_n & . & . & . & a_2 \\ . & . & & & & . \\ . & . & & & & . \\ . & . & & & & . \\ a_{n-1} & a_{n-2} & . & . & . & a_n \end{pmatrix} \text{ for } S_1$$

$$M_2 = \left(\; b_{(n+i)} b_{(n+i)-1} \cdots b_{(n+i)-n-1} \; \right) \text{ for } S_2$$

where i is the row number and $0 \leqslant i \leqslant n-1$, and M_2 is a matrix like M_1 that is shown in a vector form. The above matrices have the property that the first row is the main stream and each other subsequent row is the right shifted version of the row above by one character. We sort the above matrices according to $p = \omega_n \omega_{n-1} \ldots \omega_1$ sorting order. This order is to mean that when sorting, we first compare n-th characters and if they are equal, we compare the $(n-1)$-th characters and so on. After sorting we have the following matrices:

$$N_1 = \left(\; a_{m_i} a_{m_i-1} \cdots a_{m_i-n-1} \; \right) \text{ for } S_1$$

Top

$$N_2 = \left(\; b_{k_i} b_{k_i-1} \cdots b_{k_i-n-1} \; \right) \text{ for } S_2$$

Here, the summation in the indices is done modulus n, i.e. $(k_i+j, m_i+j) \bmod n$, for $1 \leqslant i, j \leqslant n$, $1 \leqslant k_i, m_i \leqslant n$, and for $i \neq j$ $m_i \neq m_j$ and $k_i \neq k_j$. The last columns of these matrices are the output of BWT transform, i.e. $a_{m_i+n}\ a_{m_{i+1}+n} \cdots a_{m_{i+n-1}+n}$ and $b_{k_i+n}\ b_{k_{i+1}+n} \cdots b_{k_{i+n-1}+n}$. We now prove that if these two sequences are equal then $S_1 \overset{rot}{=} S_2$. For proving this, we first note that if $N_1 = N_2$ then $S_1 \overset{rot}{=} S_2$; because S_1 is one row of N_1 and S_2 is a row of N_2 and also $N_1 = N_2$, thus, S_2 is a row of N_1, since each row in N_1 is a cyclic shift of every other row, $S_1 \overset{rot}{=} S_2$. Now we prove that if the last column of N_1 and N_2 is equal then $N_1 = N_2$. Since the last column of N_1 is equal to the last column of N_2 and the first column of each matrix is constructed from sorting the last column lexicographically, the first column of the two matrices are equal. Thus, the pairs $[a_{m_i+n}\ a_{m_i}, b_{k_i+n}\ b_{k_i}]$, $[a_{m_{i+1}+n}\ a_{m_{i+1}}, b_{k_{i+1}+n}\ b_{k_{i+1}}], \ldots$ which are the sequential pairs of S_1 and S_2, are also similar . Since each row of N_1 and N_2 starts with a pair, and rows of the two matrices are sorted, the starting pairs of their rows are also sorted, and since the pairs S_1 and S_2 are equal, so the first two columns of the two matrices are the same. And because the last columns of two matrices are similar, hence, the following triplets $[a_{m_i+n}\ a_{m_i}\ a_{m_i+1}, b_{k_i+n}\ b_{k_i}\ b_{k_i+1}]$, $[a_{m_{i+1}+n}\ a_{m_{i+1}}\ a_{m_{i+1}+1}, b_{k_{i+1}+n}\ b_{k_{i+1}}\ b_{k_{i+1}+1}], \ldots$ are the same. By using induction and the above method, we come to this conclusion that all rows in two matrices are equal one by one thus $N_1 = N_2$. Because we proved that if the two matrices N_1, N_2 are equal then $S_1 \overset{rot}{=} S_2$. Now we can conclude $S_1 \overset{rot}{=} S_2$. (It is obvious that for every input stream we have only one matrix, and each matrix has one last column, so each input stream maps into exactly one output). Hence, this method of sorting is reversible. ■

2.2. Theorem 2

Here, we prove that when sorting of two matrices are done according to $p = \omega_{k_n}\omega_{k_{n-1}} \ldots \omega_{k_1}$ (this means that in comparing two rows, we first compare k_n-th characters then k_{n-1}-th and etc.) so that

$$k_1, \ldots, k_n \in \{1, \ldots, n\}, k_n = n,$$
$$\forall i, j,\ i \neq j : k_i \neq k_j,\ \ \exists i : k_{i+1} \neq k_i + 1$$

holds, then there exist two inputs such that the last column of their BWT matrices are the same. We prove this theorem in the following three parts.

2.2.1. Part 1

Suppose the sorting order is of the form $p = \omega_n \omega_1 \omega_{k_{n-2}} \omega_{k_{n-3}} \ldots \omega_{k_1}$. Now, let's consider the following two strings:

$$S_1 = a_1 a_2 \ldots a_k B a_1 a'_2 \ldots a'_k a_k C,$$
$$S_2 = a_1 a_2 \ldots a_k C a_1 a'_2 \ldots a'_k a_k B$$

and $i \neq j : a_i \neq a_j$, $i \neq j\ : a'_i \neq a'_j$ and no substring of *B* is in *C* and vice versa. $S_1 \overset{rot}{\neq} S_2$, because of the distance between *'B'* to *'C'* is *K*+1 in S_1 and *K* in S_2. But in S_1 and S_2 strings all pairs in one string is present in the other one, and no two pairs in a stream with the same starting character has the same second character then the sorting order is determined by the first and the last column. Since the pairs of two strings are equal, therefore, the last column of the two matrices are equal; thus, we have determined two strings that will map onto the same string with the sort order p.

2.2.2. Part 2

Let's suppose that the sort order is of the form

$$p = \omega_n \omega_{k_{n-1}} \omega_{k_{n-2}} \ldots \omega_{k_1},\ k_{n-1} \neq n-1.$$

i.e. for comparison purposes we first consider the n-th column of the BWT matrix with the sort order p, then on the condition of the equivalence of the two characters in the n-th column, we consider another column other than $n-1$-th. Under this circumstance, let's assume $S_1 = a_1a_2a_3Aa_1a_4a_3B$, $S_2 = a_1a_4a_3Aa_1a_2a_3B$ and the length of *A* is equal to the length of *B* where non of a_1,a_2,a_3,a_4 are in *A*, *B* and also no substring of *A* is found in *B* and vise versa, also all the characters of *A* is higher in lexicographical order compared to each character of *B*. For example consider $S_1 = mikymake$ and $S_2 = makymike$ in Figure 2. If we construct BWT matrices for S_1 and S_2 sequences, only under the following conditions may their last column character differ in the two matrices:

I.

$$\begin{cases} a_1a_2a_3\underline{A}a_1a_4a_3B \\ a_1a_4a_3\underline{B}a_1a_2a_3A \end{cases} \text{, in the first matrix}$$

$$\begin{cases} a_1a_4a_3\underline{A}a_1a_2a_3B \\ a_1a_2a_3\underline{B}a_1a_4a_3A \end{cases} \text{, in the second matrix}$$

We can always select *A*, *B* such that the sort is determined in the underlined columns shown above, and so that *A*, *B* are in the same order in the two matrices, then the two ending characters of these rows will be similar in the two matrices.

II.

$$\begin{cases} a_3Ba_1\underline{a_2}a_3Aa_1a_4 \\ a_3Aa_1\underline{a_4}a_3Ba_1a_2 \end{cases} \text{, in the first matrix}$$

$$\begin{cases} a_3Aa_1\underline{a_2}a_3Ba_1a_4 \\ a_3Ba_1\underline{a_4}a_3Aa_1a_2 \end{cases} \text{, in the second matrix}$$

Under this circumstance, we can also select *A*, *B* such that the result of the sort is determined in the first column that has a_2, a_4 (underlined columns), and if we sort this column, then the ending characters in the two rows of the matrices are similar.

III.

$$\begin{cases} a_2a_3Aa_1a_4a_3Ba_1 \\ a_4a_3Ba_1a_2a_3Aa_1 \end{cases} \text{, in the first matrix}$$

$$\begin{cases} a_4a_3Aa_1a_2a_3Ba_1 \\ a_2a_3Ba_1a_4a_3Aa_1 \end{cases} \text{, in the second matrix}$$

Top

	S_1	S_2
	$akemikym$	$akymikem$
	$emikymak$	$emakymik$
	$ikymakem$	$ikemakym$
I.	$\underline{k}ym\underline{a}kemi$	$\underline{k}em\underline{a}kymi$
	$\underline{k}em\underline{i}kyma$	$\underline{k}ym\underline{i}kema$
II.	$\underline{m}ak\underline{e}miky$	$\underline{m}ik\underline{e}maky$
	$\underline{m}ik\underline{y}make$	$\underline{m}ak\underline{y}mike$
	$ymakemik$	$ymikemak$

Figure 2: Example of part2 for S_1=$mikymake$ and S_2=$makymike$

Under this circumstance, because the last columns of each matrices are the same, then the last characters of the two rows are similar without the need to consider how these rows are sorted. Therefore, we have found two sequences $S_1,\ S_2$ that will end up to the same output under this sorting strategy, but $S_1 \overset{rot}{\neq} S_2$.

2.2.3. *Part 3*

In this part, let's suppose that the sorting order is:

$$p = \omega_n \omega_{n-1} \ldots \omega_{n-k} \omega_{p_{n-(k+1)}} \cdots \omega_{p_1}$$

i.e. in this sorting we first compare the first k-columns and then we compare another column other than k+1. Now, let's consider two sequences $S_1 = a_1 A_1 a_2 B_1 a_1 A_2 a_2 B_2,\ S_2 = a_1 A_2 a_2 B_1 a_1 A_1\ a_2 B_2$ where $A_1,\ A_2$ are two sequences such that only their first K characters are the same and each substring with the length of K+1 in one of them exist in the other one, and also, all characters of A_1 and A_2 are upper in lexicographical order compared to each character of B_1 and B_2, and B_1 and B_2 have the properties of A_1 and A_2 respectively, and A_1, B_1, A_2, B_2 do not contain $a_1,\ a_2$. The above were our assumptions, now we start the proof; between the rows of BWT matrix the following pair of the rows in the two matrices may have dissimilar last characters:

I.

$$\begin{cases} a_1 \underline{A_1} a_2 B_1 a_1 A_2 a_2 B_2 \\ a_1 \underline{A_2} a_2 B_2 a_1 A_1 a_2 B_1 \end{cases} \quad \text{, in the first matrix}$$

$$\begin{cases} a_1 \underline{A_2} a_2 B_1 a_1 A_1 a_2 B_2 \\ a_1 \underline{A_1} a_2 B_2 a_1 A_2 a_2 B_1 \end{cases} \quad \text{, in the second matrix}$$

Under this circumstance, because the K first characters are similar in the two adjacent rows, comparison will move to another character. We can always select A_1 and A_2 such that the comparison will occur in the column containing the characters of B_1 and B_2, and therefore, the last column will be the same in the two matrices.

II.

$$\begin{cases} a_2\underline{B_2}a_1A_1a_2B_1a_1A_2 \\ a_2\underline{B_1}a_1A_2a_2B_2a_1A_1 \end{cases} \text{, in the first matrix}$$

$$\begin{cases} a_2\underline{B_1}a_1A_1a_2B_2a_1A_2 \\ a_2\underline{B_2}a_1A_2a_2B_1a_1A_1 \end{cases} \text{, in the second matrix}$$

Under this circumstance, we also can choose B_1 and B_2 such that the comparison depends on the column in A_1 and A_2, so that the sort of the last characters in the two matrices will be equal.
III.

$$\begin{cases} A_2a_2B_2a_1A_1a_2B_1a_1 \\ A_1a_2B_1a_1A_2a_2B_2a_1 \end{cases} \text{, in the first matrix}$$

$$\begin{cases} A_2a_2B_1a_1A_1a_2B_2a_1 \\ A_1a_2B_2a_1A_2a_2B_1a_1 \end{cases} \text{, in the second matrix}$$

In this case because the last column of the two rows has the same character, the similarity of the two ending columns in the two matrices dose not depend on the sort order. Therefore, we introduced the two sequences S_1 and S_2 and $S_1 \overset{rot}{\neq} S_2$ that maps into the same output under this sort order. ■
Next, we provide our improvement over BWT transform by introducing a new sorting strategy.

3. AN IMPROVEMENT OVER BWT

3.1. Method *A*

In BWT transform we put each cycle as a row of a matrix [1]. These rows will sort lexicographically and this brings similar substring at the beginning of different rotations together, and so the probability of similarity of the previous characters of these rotations (the last characters of those rotations) increases. But we note, there is a discontinuity between two rows that are different at the first character but come together. For example if *N* rows have *'A'* at the first character (*A-group*) and *M* rows have *'B'* at the first character (*B-group*), the discontinuity occurs between row *N* and row *N*+1. Row *N* that has *'A'* at the first character and its second character is lower in lexicographical order with great probability, but the second character of row *N*+1 which has *'B'* at the first character is upper in lexicographical order. Therefore, the probability of similarity of the last characters of these two rotations is low in lexicographical order. Now, let's consider that *A-group* is the same as before and *B-group* is inverted. This means that the first row of the *B-group* after these operation, is the last row of this group in the previous configuration. In this case, row *N*, the last row of *A-group* whose its second character, is lower in lexicographical order is adjacent to row N+1, the first row of *B-group* whose its second character is lower in lexicographical order too, and this increases the probability of similarity of the last characters. Also, row $N + M$ that is the last row of the *B-group* is adjacent to row $N + M$+1 which is the first row of the next group. In both of these rows, the second character probably is upper in lexicographical order. Thus, the chance of similarity of the last

File	Size	BW94	MKT	Mtd[A]	Mtd[B]
bib	111261	2.07	1.47	62	100
book1	768771	2.49	1.88	314	823
book2	610856	2.13	1.56	356	443
geo	102400	4.45	5.12	-6	82
news	377109	2.59	2.17	116	199
obj1	21504	3.98	3.05	*	*
obj2	246814	2.64	2.26	*	*
paper1	53161	2.55	1.85	-34	5
paper2	82199	2.51	1.59	-51	9
pic	513216	0.83	0.83	*	*
progc	39611	2.58	1.65	-6	-9
progl	71646	1.80	0.91	-15	-11
progp	49379	1.79	1.33	2	18
trans	93659	1.57	1.05	18	15

Table 1: Comparison of compression algorithms

characters will become larger. If we continue reversing the sort order on the basis of one-by-one groups, we will arrive at a higher rate of compression which is shown in Table 1 (the results are in the column Mtd[A] and show the reduced bytes compared to BW94).

3.2. Method *B*

In the next stage we use the fact that there are many substrings in a text that vary only in their last characters, such as:

$$\underbrace{ma\underline{k}}\,e - \underbrace{ma\underline{d}}\,e \qquad or \qquad \underbrace{abilit\underline{y}} - \underbrace{abilit\underline{i}}\,es$$

In the cyclic shifts that these different characters come at the beginning of the rows, they cause an increase in the similarity probability of their previous characters. We must next use some manipulations to bring these rows closer to each other. We denote the probability of occurrence of such characters by $P(a_i, a_j)$(the constant probability of occurrence of two substrings which only differ in their last characters a_i and a_j in any text).
If we have the values of these probabilities, we can create an alphabet order with the property that it maximizes the following sum:

$$\sum_{i=1}^{n-1} P(a_i, a_{i+1})$$

Top

where n is the number of characters in the alphabet. Now, if we use this order for sorting of the first characters of the rows and the previous order for sorting the rest of the characters, and then we use method [A], in rows which differ in the first character but they are adjacent to each other, probability of existence of the similar characters at the end of rows will reach a maximum, and method [A] will be optimum. For example, the result for the following order shown in Table 1 (the results are in the column Mtd[B] and show reduced bytes compared to BW94)

[q, c, v, o, u, a, e, i, y, r, s, t, w, h, g, l, j, p, m, n, b, f, k, d, z, x]

will improve compression ratio, because we have increased the probability that the substrings that differ in their last characters come closer to each other. In general, we can use this new order to sort all of the rows.

4. CONCLUSIONS

In this paper we considered that the sorting in BWT may result in better compression ratios; and we proved for a general class of sorting methods that the only sorting which is reversible is the one used in BWT. We also provided a new generalizable method for improving the compression results.

5. REFERENCES

[1] M. Burrows, D.J. Wheeler, “A block-sorting lossles data compression algorithm”, Digital Equipment Corporation *(SRC Research Report 124)*, 1994.

[2] S. Deorowicz,“An analysis of second step algorithm in the Burrows-Wheeler compression Algorithm”, *Software-Practice and Experience*, 2002; 32(2):99-111

[3] B. Chapin, S.R. Tate, “Higher Compression from the Burrows-Wheeler Transform by Modified Sorting”, *in Proceednig Data Compression Conference. Snowbird,UT*, 1998, p.532.

[4] P. Fenwick. “Block sorting text compression.”, *Proceeding. 19th Australasian Computer Science Conference*, pages 193-202, January 1996. ACM Press.

[5] P. Fenwick, “The Burrows-Wheeler Transform for Block Sorting Text Compression: Principle and Improvments”, *The Computer Journal*, 1996; 39(9):731-740.

[6] M. Schindler, “A Fast Block-Sorting Algorithm for lossless Data Compression”, *Proceeding of the IEEE Data Compression Conference*, 1997; 193-202.

[7] J. Ziv and A. Lempel, “A Universal Algorithm for Sequential Data Compression”, *IEEE Transactions on Information Theory*, 1977; IT-23:337-343.

Top