SRC Technical Note
2000 - 005

October 28, 2000

On variants of block-sorting compression using context from both the left and right

Mike Burrows and Li Zhang

COMPAQ

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

http://www.research.digital.com/SRC/

Abstract

The block-sorting text compression algorithm can be viewed as associating a context with each character to be compressed, and then sorting the characters on their contexts. Normally, the context associated with each character is the string to the left of the character. Recently, Ratushnyak suggested that it might be possible instead to build a context by interleaving characters taken alternately from the left and right. We show that transformations of this type are not reversible in general unless additional information is supplied. Further, the amount of additional information needed to reverse the transformation is necessarily large, and so such transformations are unlikely to be of interest as part of a compression algorithm.

1 Introduction

The first step of the block-sorting compression algorithm [1] transforms a string by sorting its characters on their left-contexts. (The left-context of character i is the string composed from characters $i-1$, $i-2$, $i-3$, ..., treating the string as cyclic.) The transformation is interesting because it is reversible and because it yields a result that is amenable to compression by straightforward algorithms.

Recently, A. Ratushnyak suggested a possible variant of the block-sorting compression algorithm that would sort the characters on contexts composed of an interleaving of the left- and right-contexts [2]. In his variant, the context for character i consists of the string composed from characters $i-1$, $i+1$, $i-2$, $i+2$, $i-3$, $i+3$ etc. Context information from both sides is likely to be a better predictor for a character than context information from one side, and indeed this suggestion results in significantly better compression of the result of the transformation.

Ratushnyak provides a worked example demonstrating how one can reverse the transformation on a 13-character string. The techniques he uses are interesting and may lead to further insights. However, he gives neither a complete algorithm for reversing his transformation nor a proof that any such algorithm exists. This left open the question of whether his suggestion could indeed form the basis of a useful compression algorithm.

2 Ratushnyak's transformation is not reversible

At the suggestion of Jim Saxe, we applied Ratushnyak's algorithm to short binary strings and we soon found collisions. That is, we found inputs that are distinct under rotation yet which are mapped to the same string by Ratushnyak's transformation. The shortest colliding strings have length 6. An example colliding pair is 110100 and 101100, which both map to 110100 under Ratushnyak's transformation. As the length of the strings increases, the number of collisions quickly becomes large.

3 A more general negative result

The presence of collisions in Ratushnyak's transformation does not necessarily undermine its use in a compression algorithm since we may add additional bits to resolve ambiguity. Indeed, in the original block-sorting algorithm, we have to add a $\log n$-bit string in order to distinguish the n possible rotations of the decoded string.

In general, we have to use at least $\log M$ bits to resolve an M-way collision. In the following, we construct a set of c^n strings of length n, where $c > 1$, all of which are transformed to the same string under Ratushnyak's algorithm. In other words, we need linearly many additional bits, $n \log c$ of them, to resolve the ambiguity.

For an even number $n > 0$, consider a string s with the following properties:

- it consists of $\frac{3}{2}n$ 0's and n 1's;
- it starts with 0 and ends with 1;
- it does not contain two consecutive 1's; and
- it does not contain three consecutive 0's.

Now, we claim that:

1. **Any such string s is transformed to the string $1 \cdots 10 \cdots 0$.**

   For each 0 in s, its context starts with either 10 (when it appears first in two consecutive 0's), 01 (when it appears second in two consecutive 0's), or 11 (when it is sandwiched by two 1's). For each 1, since it is always between two 0's, its context always starts with 00. Thus, all the 1's have contexts that sort before the 0's. Therefore, s is transformed to the string $1 \cdots 10 \cdots 0$.

2. **The number of such strings is $\Theta(2^n/\sqrt{n})$.**

   We can construct such strings as follows. Consider a permutation of $n/2$ a's and $n/2$ b's. We first insert a 1 after each letter in the string. Then, we replace each a with 0 and each b with 00. It is easy to verify that strings constructed this way satisfy the required constraints. The number of such permutations is $\binom{n}{n/2} = \Theta(2^n/\sqrt{n})$.

We thus have constructed $\Theta(2^n/\sqrt{n})$ strings with length $\frac{5}{2}n$ so that they are all transformed to the same string $1 \cdots 10 \cdots 0$ under the algorithm where the context of each character starts with its two neighboring characters. Or equivalently, we need to use about $2/5 = 0.4$ additional bits, per bit in the original string, to resolve the ambiguity. A similar, but more complex construction shows that the number of additional bits required can be as high as 0.4057.

Notice that this argument applies not only to Ratushnyak's transformation but also to any similar transformation in which the context of a character starts with its two neighboring characters, in either order.

4 Acknowledgement

We would like to thank Lyle Ramshaw for many useful comments on a draft of this note.

References

[1] M. Burrows and D.J. Wheeler. “A Block-sorting Lossless Data Compression Algorithm”. Research report 124. 10th May 1994. Digital Equipment Corporation Systems Research Center, 130 Lytton Ave, Palo Alto, CA USA. http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-124.html

[2] A. Ratushnyak. “Sorting the matrix of two-sided contexts”. Usenet news article `<8pt96o$nua$1@nnrp1.deja.com>` posted to comp.compression. 15 September 2000. Also available at http://geocities.com/eri32/slrm.htm and http://artest1.tripod.com/slrm.htm