Burrows Wheeler - Alternatives to Move to Front

Dr Peter Fenwick
Dr Mark Titchener
Michelle Lorenz

Department of Computer Science, The University of Auckland,
Private Bag 92019, Auckland, New Zealand
p.fenwick@auckland.ac.nz
mark@tcode.tcs.auckland.ac.nz
mlor009@ec.auckland.ac.nz

1 Abstract

The Burrows Wheeler Transform, first published in 1994 [2], is a relatively new approach to text compression and has already proven to produce excellent results. However, there has been much research directed towards improving the efficiency of the Move to Front algorithm[7, 1, 4] with varying degrees of complexity. This paper examines a relatively simple technique using a cached modeling system and achieves very promising results. Information loss during the Burrows Wheeler Transform is also examined using Deterministic Information Theory measuring techniques. This provides an interesting insight into what is actually occurring during the complete Burrows Wheeler Transform and can confirm whether manipulating the MTF algorithm is the correct approach to refining an already successful compression scheme

2 Introduction

Unlike conventional statistical compression schemes, the Burrows Wheeler transform itself does not compress the data input stream but rather converts it into a more amenable format for another compression method. The permuted output from the BWT has the property of grouping characters together. Conventionally this is then passed to a MTF algorithm with the intention of further reducing the data before encoding it with a variable length coder.

The MTF coder produces a highly skewed symbol distribution, with 0 consistently prominent and the higher positions/symbols usually not occurring more than twice. Much research has been focused on refining the MTF, most based on controlling when to move the symbol

to the front of the stack. This paper takes a different approach. Instead of attempting to improve the MTF algorithm it examines two relatively simple methods, based on models suggested by Fenwick [4] and Balkenhol [1] for manipulating the permuted data to produce better probability estimates for the variable length coder. These are discussed in detail in Section 4.

In order to employ an efficient alternate method a clearer insight of the actual information content and/or loss after the Burrows Wheeler and MTF algorithms need to be established. Titchener has developed a highly useful tool using deterministic information theory, first published in [6], for measuring information over a finite string. Deterministic IT can be used to calculated the complexity, information and entropy associated with any finite string and will be referred to as T-complexity, T-information and T-entropy respectively to disallow any confusion with classical measurements.

The information measure is based on recursively decomposing a finite string into a set of highly synchronous codewords. The complexity of the string, the T-Augmentation, is then represented by the number of steps required to recursively construct the string from its alphabet and is measured in *taugs*.

The T-augmentation is effectively a string production rule for the longest strings and the number of applications required to construct these strings.

The *T-information* of a string is the deterministic information content of a string s referred to as $I_{det}(s)$. It is calculated as the inverse logarithmic integral of the string corresponding to the deterministic complexity $C_{det}(s)$. That is

$$I_{det}(li^{-1}(C_{det}(s)) \quad [6]$$

T-information is measured in symbols/per symbol known as nats.

Furthermore Titchener has also defined a third information measure, *T-entropy*. T-entropy is represented as

$$T_E = \triangle T_I / \triangle L \quad [6]$$

It is the rate of change of T-information T_I along a string L, and will be used to measure the average T-entropy over a file.

In order to analyze the string the DIT software *tcalc*[6] will be used to produce the T-complexity, T-augmentation, T-entropy on selected files during the various stages of the complete compression process.

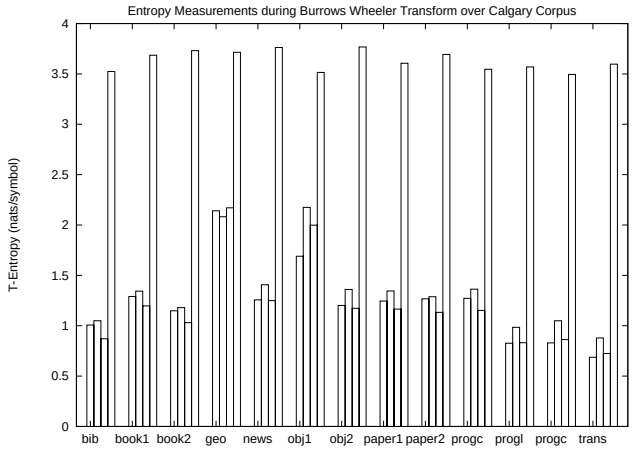

Figure 3.1: T-entropy

3 Information over the Burrows Wheeler Transform

Entropy measurements were taken over a standard BWT compression algorithm.

Figure 3.1 shows the T-entropy after each of the four steps, over selected files from the Calgary Corpus. For each file, the first (left-hand) bar shows the T-entropy for the input file, in nats per symbol. As a value of 4.0 implies chaotic data, these values should be doubled (actually times 7/4 for 7-bit ASCII data) to get the more usual measure of bits/byte, or bits/character.

The second bar shows the T-entropy after the Burrows Wheeler transform. For most files the T-entropy for the transformed file is slightly higher than for the unpermuted input. (The exceptional file, geo from the Calgary corpus, contains binary data with other unusual characteristics.)

The third bar of each group shows that the T-entropy is usually reduced by the Move To Front stage.

The fourth bar shows the information of the final compressed output. Given that an ideal compressor gives completely random output and that completely chaotic data has a T-entropy of 4 nats/symbol, a T-entropy of 3.6 implies that a possible 10% gain in compression might be still available from an optimum compressor. The heights of these bars therefore indicate the quality of the final compression.

While there are only minor variations in T-entropy values in the three files prior to the final encoding, it would seen that BWT tends to add extra information and after applying the MTF this information is then lost again. Under further experimentation these characteristics

were verified using files of consistent size, type and alphabet size, confirming that these property generally hold true for any files.

Despite no variation in file size, there is a consistent decrease in entropy between the BWT and MTF stages. While the BWT adds small quantities of information, the MTF efficiently utilizes the structure of the BWT, producing an output with significantly lower overall information content, making it ideal for compression. An explanation for these traits is offered in section 5. Because at the MTF stage the data is completely permuted and contains no contextual information, this verifies it is feasible for efforts to be directed towards improving this step.

4 Dual modeling of Move to Front data stream

Applying the MTF algorithm after the BWT creates a highly skewed symbol distribution bearing zero similarities to the original file. Effectively MTF ranks the symbols such that 0 will have a very high statistical value as will 1, but this will rapidly decrease. Figure 4.1 compares the MTF symbol distribution against the original alphabet over the file *book1* taken from the Calgary corpus. Due to distortion created by the high values of the front symbols, the logarithmic frequencies were taken. The distribution of the original file displays an approximate constant inversely proportional relationship between logarithmic frequency and ranked symbol. However, the MTF can almost be modeled by an inverse exponential function. It has a few highly frequent symbols, but the majority occur only a few times.

Despite this distributive property, the entire MTF input stream is conventionally represented in one model and usually encoded with an arithmetic encoder. However, this symbol distribution could be better utilized by separating the alphabet into two models before the final encoding.

One method suggested by Fenwick relies on a cache system [4], where the most probable symbols are stored in a prominent foreground model, and the bulk remaining symbols stored in a larger background model.

Theoretically this approach will allow for superior compression as better probability estimates can be achieved because the context of the highly probable symbols in the foreground model will not be distorted by the more skewed background model.

In a dual modeled representation of the MTF data, successful compression is assured by providing the decoder a means of knowing when to switch between models. Because most

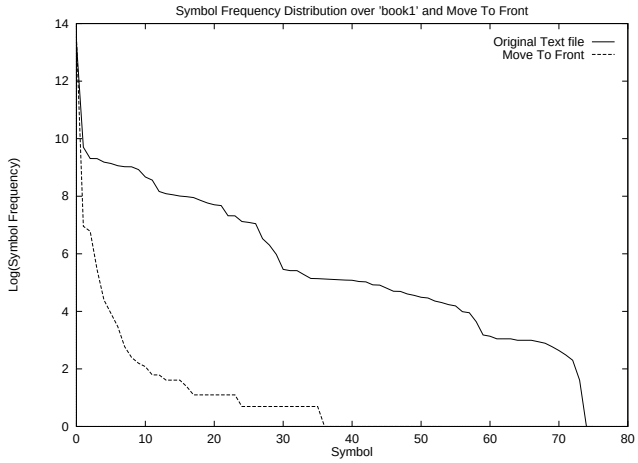

Figure 4.1: Symbol Distribution on original file and after the MTF Transform

symbols should be encountered in the cached foreground model, both the encoder and decoder assume its state by default. If a background symbol is encountered the encoder emits a special *ESCAPE* symbol from the foreground model, informing the decoder to switch models before then encoding the symbol in the background model.

Ideally as with PPM [5]the probability of the *ESCAPE* will be tailored to predict an *UNKNOWN* or background model symbol. For demonstration purposes this symbol has been represented as an extra addition to the cached alphabet. Its probability is determined by its frequency.

A similar approach has been suggested by Balkenhol[1] where encoded in the original ASCII alphabet instead of the conventional MTF. In addition further enhancements are made to the cache with provisions to ensure a symbol is only moved to the very front and assigned zero, if the adjacent symbols are the same. This method has the drawback of increased computational insensitivity, as the original ASCII text of the background model must be computed and stored additionally in memory throughout the program.

Figure 4.2 shows the typical results when varying the size of the foreground model for both methods on the file *book1*. The x-axis demonstrates the foreground model size while the y-axis displays the bits per byte performance.

With the exception of *geo* all other files rendered similar behavioral patterns. It becomes clear that using a cache system approach yields better compression. However, the optimal model size varies for each file and method. Preliminary statistics suggest the optimal cache size is a function of the file's alphabet size and type.

It appears the straight MTF cache delivers slightly better results than the latter method

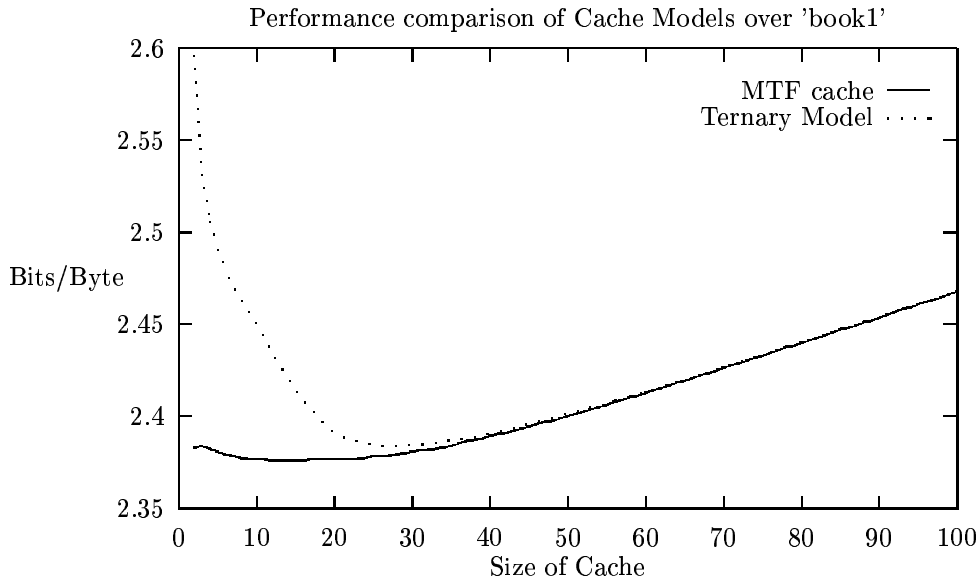

Figure 4.2: Bits per byte performance on varying foreground model size

suggesting the information content of the ASCII background model is more skewed than the MTF data.

A compression performance summary for both methods at their optimal points is displayed in table 4.2 and compared against the BWT results achieved by Wirth using PPM. Because no consistent correlation between optimal cache size and performance was evident, it is envisioned that the encoder will have to make an initial pass over the file to determine an ideal cache size to use and transmit this additionally to the decoder.

The results from Table 4.2 allow for two definite conclusions. Firstly, the straight cached modeling system consistently performs equal or better than the ternary modeling technique. Considering the complexity involved with the ternary model, it would be logical to focus further refinement on the bare cached model. Secondly, in comparison with Wirth's results, the overall average reveals an approximate equal if not slightly superior outcome. This is very promising for a relatively simple technique that has much potential for further refinement, such as efficient coding of the *ESCAPE* symbol, suggesting improvements are viable and faster than comparable methods.

File	MTF cache	Ternary Cache	Wirth
bib	1.953	1.965	1.998
book1	2.376	2.384	2.330
book2	2.033	2.044	2.012
geo	4.593	4.578	4.390
news	2.488	2.506	2.487
obj1	3.850	3.906	3.811
obj2	2.470	2.487	2.514
paper1	2.460	2.480	2.492
paper2	2.411	2.425	2.424
progc	2.499	2.529	2.518
pic	0.758	0.765	0.743
progl	1.717	1.730	1.763
progp	1.718	1.739	1.792
trans	1.502	1.518	1.622
Average	2.345	2.361	2.350

Table 4.2: Compression comparison of cached models against Wirth

5 Analysis of Information

This section focuses on understanding the behavior of the information contents particularly after the BWT and MTF stage.

Section 3 already established the move-to-front algorithm undergoes information loss despite size remaining unchanged. This is expected as the algorithm generally will reduce the alphabet size and produce a very skewed symbol probability distribution.

Curiously the BWT consistently tends to gain slight quantities of information, despite being approximately the same size. To attempt to understand this somewhat counter-intuitive trait, the actual structure of the file needs to be considered as well as how the information content varies throughout. A 50Kb text block was taken from *book1* and analyzed more closely. Figure 5.1 models the continuous T-entropy along the BWT output. In comparison the T-entropy across the original file is also included as well as the output from the MTF. T-entropy is shown in the y-axis, while the x-axis shows the file position. Immediately the behavioral difference between the BWT and original file is evident. The original file has a relatively consistent T-Entropy, while the T-entropy from the BWT fluctuates continually [3], a trait reflected by the MTF. This property can be explained when considering the actual structure of BWT. Because BWT clumps characters together a typical string might

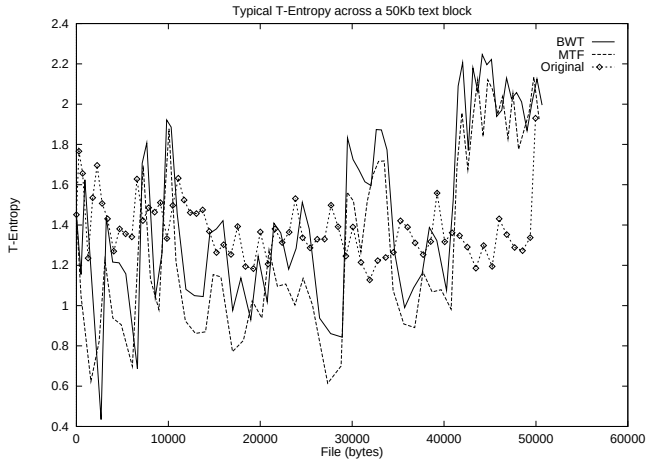

Figure 5.1: View of information rate along a singular transformation. Due to the permutations there is no relationship/pattern between the original and transformed files

contain:

$$efeffeefeeeeeeeeeeeeeeeeeeeeeeeeaeaeaaaaaaaaaaaaa$$
$$\uparrow$$

Imagine the entropy at the point detonated by the arrow. The encoder has already seen numerous e's and with each successive e the entropy drops, then suddenly an unexpected symbol, a, is encountered causing the T-entropy to increase dramatically due to the addition of the previously unseen symbol. Then with each successive a the entropy steadily drops until the next (new) symbol is encountered. Another property that becomes evident is the MTF file approximately mirrors the BWT, but at a lower information level. This is consistent with the aforementioned theory. Consider the following typical two sample output streams from a BWT and MTF respectively:

$$aaaaaaaababababbbbbbbbbbbb, b, b, , , , , , , , , , , , zzzzzzzzzzzzzzz$$
$$\uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow$$
$$000000001111100000000000211110000000000300000000000000$$
$$\uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow$$

It is observed that the MTF algorithm will introduce a new symbol to its alphabet at approximately the same point as the BWT. The lower information value is accounted for

by a generally smaller alphabet and a skewed distribution, i.e. a higher value of 0's and 1's than other symbols.

5.1 Summary

Using the information measurement techniques proposed by Titchener [6] a clearer understanding of informational changes during BWT compression has been obtained, specifically information loss during the BWT and MTF stages. While the BWT adds small quantities of information, the MTF efficiently utilizes the structure of the BWT. The behavioral content along both streams produces a similar pattern with the MTF producing a significantly lower overall information content, making it very suitable for compression. The mirrored behavior of T-entropy along both streams implies similarities in structure and indicate both, especially the MTF contain little contextual information.

6 Conclusions

By utilizing the information measurements tools developed by Titchener [6] a clearer understanding of informational changes over BWT has been established. Most importantly MTF loses information making it amenable for compression, however it also contains little contextual structure. Therefore, efforts have been focused on developing a better representation of the MTF stream. Two methods were considered. The best method, proposed by Fenwick, employed a simple cache system where the prominent symbols were encoded in the foreground model and the remaining symbols encoded in a separate background model, allowing for better probability estimates. The cached system fractionally out-performed results achieved by Wirth and holds promise for further improvements.

References

[1] Bernhard Balkenhol and Yuri M. Shtarkov. One attempt of a compression algorithm using the bwt. Technical report, Bielefeld University, Postfach 100 131, 44501 Bielefeld, Germany.

[2] M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm. Src research report 124, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301, May 1994.

[3] Sebastian Deorowicz. Second step algorithms in the burrows-wheeler compression algorithm. *Software-Practice and Experience: 2002; 32:99-111*, 2001.

[4] Dr Peter Fenwick. Block sorting text compression - final report. Technical Report 130 ISSN 1173-3500, University of Auckland, Department of Computer Science, Auckland, 1996.

[5] Timothy C. Bell Ian Witten, Alistair Moffat. *Managing Gigabytes*. Morgan Kaufmann Publishers, 1999.

[6] Dr Mark R Titchener. Deterministic computation of string complexity, information and entropy. *Third International Conference on Information Theoretic Approaches to Logic, Language and Computation*, June 16-19 1998. Histou, Taiwan.

[7] Anthony Ian Wirth. Symbol-driven compression of burrows wheeler transformed text. Master's thesis, The University of Melbourne, 2000.